A Software Approach to Enhancing Quality of Service in Internet Commerce

Yussuf N. Abu-Shaaban

Doctor of Philosophy
University of Edinburgh
2004

Abstract

In many e-commerce systems preserving Quality of Service (QoS) is crucial to keep a competitive edge. Poor QoS translates into poor system resource utilisation, customer dissatisfaction and profit loss. One way to ensure that QoS requirements are met is to use more hardware. However, adopting a software approach could be both cheaper and easier to manage. A combination of software solutions could be effective in aiding the process of making QoS-oriented decisions in different stages of the e-commerce system's life-cycle.

A major contribution of this thesis is a general-purpose Internet Commerce performance benchmarking tool, ECBench. It is a design-stage software solution that enables the benchmarking of different e-commerce technologies and techniques. Using ECBench, an e-commerce application developer can evaluate competitive technologies. The criteria of choice would be how well they meet a set of QoS system requirements. ECBench is based on the TPC-W standard specification, thus providing a realistic model of an Internet Commerce system. A comprehensive set of metrics and experiment design facilities are provided in the tool to enable a detailed performance analysis of the e-commerce technologies and techniques which are benchmarked. To demonstrate the use of the ECBench, PHP and Java Servlets application server technologies have been incorporated into the tool and experiments have been designed and performed to investigate and compare the scalability of the two technologies.

Another major contribution of the thesis is a novel Cost-Based Admission Control solution (CBAC) to preserve QoS in Internet Commerce systems. CBAC is a dynamic software solution which uses a congestion control mechanism to maintain QoS while the system is online. Rather than rejecting customer requests in a high-load situation, a discount-charge model which is sensitive to system current load and navigational structure is used to encourage customers to postpone their requests. A scheduling mechanism with load forecasting is used to schedule user requests in more lightly loaded time periods. The effect of CBAC on QoS has been investigated by benchmarking it on ECBench. It has been found that the use of CBAC at high load achieves higher profit, better utilisation of system resources and service times competitive with those which are achievable during lightly loaded periods. Throughput is sustained at reasonable levels and request failure at high load is dramatically reduced.

Acknowledgements

Firstly, *Alhamdllah*, all praise is for God.

Then,

I would like to thank Dr. Jane Hillston for being an excellent supervisor. Thanks Jane for your valuable advice, generous support and giving me the time whenever I needed it. Working with you was an enjoyable and rewarding experience. Thanks Jane for everything.

Many thanks to my parents, father Nabeel and mother Iman. As always, your endless love, unconditional care and support are driving forces in my life. Without your direction, advice and patience, doing this Ph.D. would have been impossible.

Many thanks to my wife Rania. Thanks Rania for your great love and massive support. Thanks for being an excellent listener and for caring. Thanks for suspending your studies to join me in Edinburgh. I would like also to thank my son Nabeel for being a huge inspiration in the last 16 months. Thanks also for being an effective alarm clock without knowing it.

I would like also to thank my sisters Nadine and Sandra, and my brother Mohammed. Your love and support have been great throughout the years.

Thanks to all my friends and the rest of my family for their love and support.

I would like also to thank the Engineering and Physical Sciences Research Council (EPSRC) for their financial funding, studentship 00317428, through the Dependability Interdisciplinary Research Collaboration (DIRC) project.

Finally, I would like to thank all members of staff at the school of Informatics including Mr. Stuart Anderson and Dr. Stephen Gilmore for their help and support.

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

Chapter 3 of this thesis is based on [1], accepted and presented at the 18th UK Performance Engineering Workshop 2002 (UKPEW2002).

Chapter 4 of this thesis is based on [2], accepted and presented at the 10th International Conference on Analytical and Stochastic Modelling Techniques and Applications 2003 (ASMTA2003).

(Yussuf N. Abu-Shaaban)

[redacted]

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The size and value of Business-to-Consumer (B2C) Internet Commerce transactions have been increasing in recent years. An analysis performed by eMarketer [5] indicates that, in 2003, the UK had 15.7 million online buyers spending an average €745 and generating a total revenue of €11.7 billion. Across Western Europe, €33.4 billion was generated from B2C e-commerce sales. According to the US Census Bureau [6], retail e-commerce sales for the second quarter of 2004 was \$15.7 billion, an increase of 23.1% from the second quarter of 2003. To cope with such increasing demand, e-commerce system developers and administrators need to consider Quality of Service (QoS) enhancement in various stages of the system's life-cycle.
This thesis provides two major contributions to enhancing Internet Commerce QoS. The first is a general-purpose Internet Commerce performance benchmarking tool ECBench, which enables making QoS-oriented decisions at early stages of the system design process. The second contribution of the thesis is a novel Cost-based Admission Control solution (CBAC) which can be applied to Internet Commerce systems, that are online and fully operating, to preserve QoS. An overview of the thesis content is provided in the rest of this chapter.

Central to the work presented in this thesis is the performance modelling of e-commerce systems. Definitions and background information on e-commerce are presented in Chapter 2. Various definitions of the term *e-commerce* are presented and a broad definition of e-commerce is adopted which includes different known types of e-commerce and the organisational and commercial effects of such systems. The chapter includes a review of different types of e-commerce systems emphasising that Internet Commerce retailing is the domain considered in this thesis. Background on the performance modelling of Internet Commerce systems is also provided in this chapter. This includes describing the business model, the architecture, workload characterisation and QoS metrics of such systems.

The first major contribution of the thesis, the ECBench tool, is described in Chapter 3. ECBench facilitates the performance benchmarking of different e-commerce technologies and techniques. It can aid an Internet Commerce application developer, in the system design stage, to choose between alternative technologies to meet a set of QoS requirements. A researcher can use ECBench to investigate new techniques to enhance Internet Commerce QoS. Based on the standard TPC-W specification [3], ECBench provides a realistic emulation of an Internet Commerce retailing system. The tool's modular design enables its extension to incorporate new technologies and techniques to be benchmarked. A comprehensive set of Internet Commerce QoS metrics can be evaluated using ECBench including service time, throughput, profit per second and system resource utilisation. The tool also includes a set of experiment design features allowing the user to specify and run a set of experiments. Results obtained can be analysed and presented graphically by ECBench.
The use of ECBench to comparatively benchmark PHP and Java Servlets application server technologies is described in Chapter 4. Background on PHP and Java Servlets is provided before discussing the incorporation of the technologies into ECBench. The chapter also includes a description of the experiments performed using ECBench and the results obtained when investigating the scalability of the two technologies.

The second major contribution of the thesis, CBAC, is presented in Chapter 5. CBAC is a novel cost-based admission control software approach to enhance Internet Commerce QoS by controlling system congestion. Rather than rejecting customer requests to reduce system congestion, CBAC is based on a discount-charge pricing model to encourage customers to postpone their requests when the system is approaching a congested state. The pricing model is dependent on system current load and navigational structure. A scheduling mechanism with load forecasting is used to schedule user requests in more lightly loaded time intervals in return for getting a discount. A customer is charged if they decide to pursue a request in a heavily loaded situation. Chapter 5 provides a review of techniques used and proposed in the literature for QoS enhancement while the system is online. The chapter also includes a description of CBAC's pricing model and how it is affected by current system load and navigational structure. Due to CBAC's sensitivity to system navigational structure, an analysis of the Internet Commerce dependent service model is provided before describing CBAC's Navigational Model (CBAC-NM). CBAC's scheduling mechanism and the use of load forecasting in request scheduling are also described, before discussing CBAC deployment and applicability.

CBAC provides a comprehensive framework to enhance the QoS metrics vital to Internet Commerce system performance. The distribution of high load to less loaded future time intervals reduces request service time and increases system utilisation. Throughput is also sustained and request failure is reduced. The charging element of CBAC, the abandoning of request rejection and increasing customer satisfaction by offering discounts and better service times all contribute to maximising profit. Work done on investigating CBAC performance is described in Chapter 6. The extension of ECBench to benchmark CBAC is discussed before describing experiments performed to analyse the performance gains that can be achieved by adopting CBAC and how CBAC parameters can be tuned to optimise different QoS metrics. Conclusions and directions for potential future work are described in Chapter 7.

Chapter 2

Background

2.1 Introduction

In this chapter, a general background is provided on e-commerce and its performance modelling. Different definitions of e-commerce and e-business are examined in Section 2.2. This is followed in Section 2.3 by a review of different types of e-commerce models and systems. Elements realising the performance modelling of Internet Commerce systems are described in Section 2.4 including Internet Commerce business model, system architecture, workload characterisation and metrics needed to evaluate Internet Commerce QoS.

2.2 E-commerce Definition

There is no agreed definition for e-commerce. Some adopt a specific definition and think of e-commerce as an Internet phenomenon. They argue that e-commerce is commerce enabled by Internet technologies. Others argue that e-commerce systems such as Electronic Data Interchange (EDI) were used years before the Internet era and thus specifying e-commerce systems as only Internet-based systems is not accurate. Whiteley [7] provides a general (rather technical) definition of e-commerce: *'Formulating commercial transactions at a site remote from the trading partner and then using electronic communications to execute that transaction'*. Another definition is provided in a European Commission introductory paper to e-commerce [8]: *'any form of business transaction in which the parties interact electronically rather than by physical exchanges or direct physical contact'*. The technical and business changes required when e-commerce is used are captured in a more comprehensive definition of e-commerce provided by the Organisation of Economic Cooperation and Development (OECD) [9]: *'Electronic Commerce refers generally to all forms of commercial transactions involving both*

organisations and individuals that are based upon the electronic processing and transmission of data, including text, sound and visual images. It also refers to the effects that the electronic exchange of commercial information may have on the institutions and processes that support and govern commercial activities. These include organisational management, commercial negotiations and contracts, legal and regulatory frameworks, financial settlement arrangements, and taxation, among many others'.
All of the above definitions imply the broadness of e-commerce in including Internet Commerce, Business-to-Business (B2B), etc. Based on that, a classification of e-commerce applications is provided in Section 2.3. A difference between the above definitions is whether e-commerce is only restricted to the electronic buying and selling or it also includes the adaptations required in the business model as emphasised in OECD definition of e-commerce quoted above. Throughout this thesis, the broader meaning of e-commerce implied in OECD's definition is adopted.

E-business is another term that is closely related to e-commerce. Again, different people use e-business to refer to different things. Some consider it to be only specific to B2B applications. Others treat e-business as synonymous with the broad definition of e-commerce described above and that is what is used in this thesis.

2.3 Classification of E-commerce Systems

A common classification of e-commerce systems is based on the parties interacting in the system such as businesses, consumers and governments [10] [8]. This includes Business-to-Consumer (B2C) which represents a main commercial use of the Internet. Another class of e-commerce is Business-to-Business (B2B) which involves all transactions made by a company and its suppliers. Consumer-to-consumer (C2C) and e-government in the form of Government-to-Business and Government-to-Citizen transactions could be considered as other classes of e-commerce.
In the rest of this section, a review of B2C and B2B is given as they are the most common forms of e-commerce. In Section 2.3.1, B2B is discussed; this is followed in Section 2.3.2 by a review of B2C business models.

2.3.1 B2B E-commerce

B2B e-commerce refers to the commercial transactions between organisations. Electronic Data Interchange (EDI) systems are the common form of B2B e-commerce. EDI systems provide a standardised way for coding trade transactions, so that they can be communicated across organisations, e.g. linking a large supermarket chain or an aircraft manufacturer with their suppliers. A traditional EDI system is based on a Value-Added Network (VAN) which is a closed network that organisations interacting in the system can access. Parties involved in EDI transactions over a VAN must agree on a standardised way for transferring electronic documents. The format of such documents must be agreed at the start of the business relation. Recently, the use of Internet-based EDI systems minimises the effort required to standarise business partners transactions as communication is achieved over a common medium which is the World Wide Web (WWW). In addition, using the expressive eXtensible Markup Language (XML) eases the process of defining electronic documents exchanged between trading partners.

2.3.2 B2C E-commerce

The medium of interaction between business and consumer in B2C e-commerce systems is usually the Internet. For that, Internet Commerce is the terminology that is usually used to describe such systems. Whiteley [7] describes Internet Commerce as the commercial use of the Internet where information and technologies can be used to allow Internet consumer purchasing. Goods can be delivered by post or delivery can be electronic in the case of electronic materials. Several types of Internet Commerce applications can be identified [11] [12] [7]. The most common of these are described below:

- Retailing, the most widely known application of Internet Commerce. Amazon.com [13] and etoys.com [14] are examples of Internet Commerce retailing sites. In such systems, online catalogues are used by merchants to offer the products they are selling. The system includes a set of services which customers can request to browse merchants' catalogues and place orders. A detailed description of the Internet Commerce retailing business model and its functions is provided in Section 2.4.3.

- Auction sites are Internet Commerce applications which enable sellers and bidders to trade online. A seller can post to the site an item for sale together with a minimum price and deadline to close the auction. A bidder can search the site for specific items, view current bids on an item and provide a bid.

An example of a major auction site is ebay [15]. *Reverse Auction* is another auction model in which a buyer requests the purchasing of an item with a maximum price. Sellers bid to provide the lowest price.

- Portals are sites that contain many resources on particular topic(s). Shopping pages are included in portals to link customers to thousands of merchants that offer products related to the portal's topic. One possible source of revenue for portal owners is to charge merchants for each click from the portal pages to the merchant site. yahoo.com [16] is an example of a portal site.

- Electronic Markets (EM) which represent the use of information and communication technology to present offers available in a market segment so that the purchaser can compare prices before making a purchase decision [7]. Several models can be used in EM sites [11] including *Name-Your-Price Model*, *Comparison-Pricing Model* and *Demand-Sensitive-Pricing Model*.

Throughout this thesis, only B2C e-commerce is considered, in particular the retailing Internet Commerce model. The terms e-commerce and Internet Commerce will be treated as synonymous with Internet Commerce retailing.

2.4 Internet Commerce Performance Modelling

To model the performance of an Internet Commerce system, the following is required: the business model of the system, its architecture, characteristics of its workload and relevant performance metrics. In this section, a detailed description is given of how performance modelling of Internet Commerce can be realised. General performance modelling concepts are discussed in Section 2.4.1. A framework for the quantitative performance analysis of e-commerce systems devised by Menasce and Almeida [10] is described in Section 2.4.2. The business model of Internet Commerce retailing is described in Section 2.4.3. The architecture of a typical Internet Commerce system is provided in Section 2.4.4. Work done on web server and Internet Commerce workload characterisation is reviewed in Section 2.4.5. Finally, in Section 2.4.6, QoS of Internet Commerce systems and metrics used to measure its performance are discussed.

2.4.1 Performance Modelling Concepts

Performance evaluation of systems can be achieved through direct system measurements. However, gaining access to systems for performance evaluation is not always feasible especially in commercial systems. An alternative approach is to use a model of the system to evaluate its performance. This is a more flexible approach where the right level of model abstraction can be used so that unimportant system details are not included. The model should capture system characteristics that affect performance including system workload and architecture. A set of performance metrics should also be specified. Once the system model is constructed and its parameters are specified, evaluation of the model can be done using simulation or analytical methods. Emulation is a type of model simulation [17] where the system model is used to construct a real system with the characteristics specified in the model. Measurements are then taken for different model metrics to investigate system performance under different alternative model parameter sets.

Performance modelling can be used for system capacity management and planning. Jain [17] defines *capacity management* as *the problem of ensuring that the currently available computing resources are used to provide the highest performance* while *capacity planning* is defined as *ensuring that adequate computer resources will be available to meet the future workload demand in a cost-effective manner while meeting the performance objectives.* Ensuring that QoS requirements are met at present is the purpose of capacity management while capacity planning is concerned with QoS guarantees in the future. Benchmarking tools can be used to compare the effectiveness of alternative technologies and techniques in aiding capacity management and planning. Such benchmarking tools could be based on performance model emulation of the system for which capacity management and planning is required.

2.4.2 Menasce's E-business Reference Model

Menasce and Almeida in [10] devised a methodology for the capacity planning of e-commerce systems. The methodology is based on an e-business reference model that can be used as a framework for the quantitative performance analysis of e-commerce systems. As shown in Figure 2.1, the model consists of four layers grouped into two parts. The upper part includes the business and function models of the e-commerce system while the lower part is concerned with modelling system workload and resources. The business model specifies the type of service offered by the e-commerce system and the properties of the service market. The process of delivering the service using the functions available in the system and the navigation between them is captured by the function model. The Customer and Resource Models determine the demand that system workload imposes on

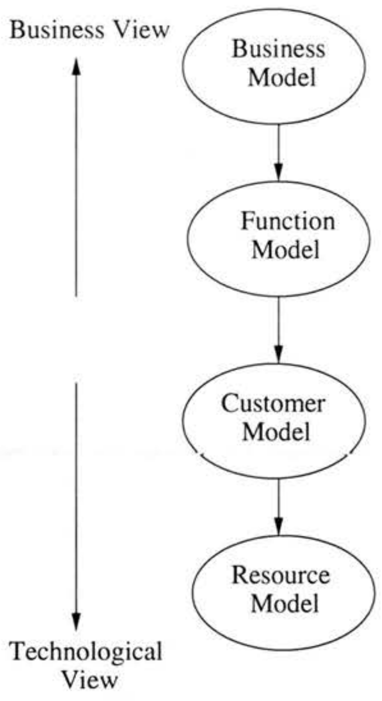


Figure 2.1: Menasce's Reference Model for E-business

the system's resources. Factors used to characterise system's workload include customer navigational pattern and intensity of requests.

2.4.3 Internet Commerce Retailing Model

Retailing applications follow the *shopfront model* in which transaction processing, information storage, security and payment are combined to enable the interaction between merchants and customers online [11]. Merchants provide online catalogues of products they are offering. A customer uses services such as Search Request, Search Result, Product Details, Best Sellers and New Products to browse the merchant's product catalogue. A shopping cart is a central service in this model which customers use to accumulate a selection of items from the merchant's product catalogue. Ordering services include Customer Registration, Buy Request, Buy Confirm, Order Enquiry and Order Display. A customer uses those ordering services to provide personal details, order items in the shopping cart and enquire about the status of orders placed. Ordering services involve online payment and authentication.

2.4.4 Internet Commerce System Architecture

Figure 2.2 is a block diagram of a simple Internet Commerce system architecture. A typical Internet Commerce site is based on a 3-tier system containing a Web

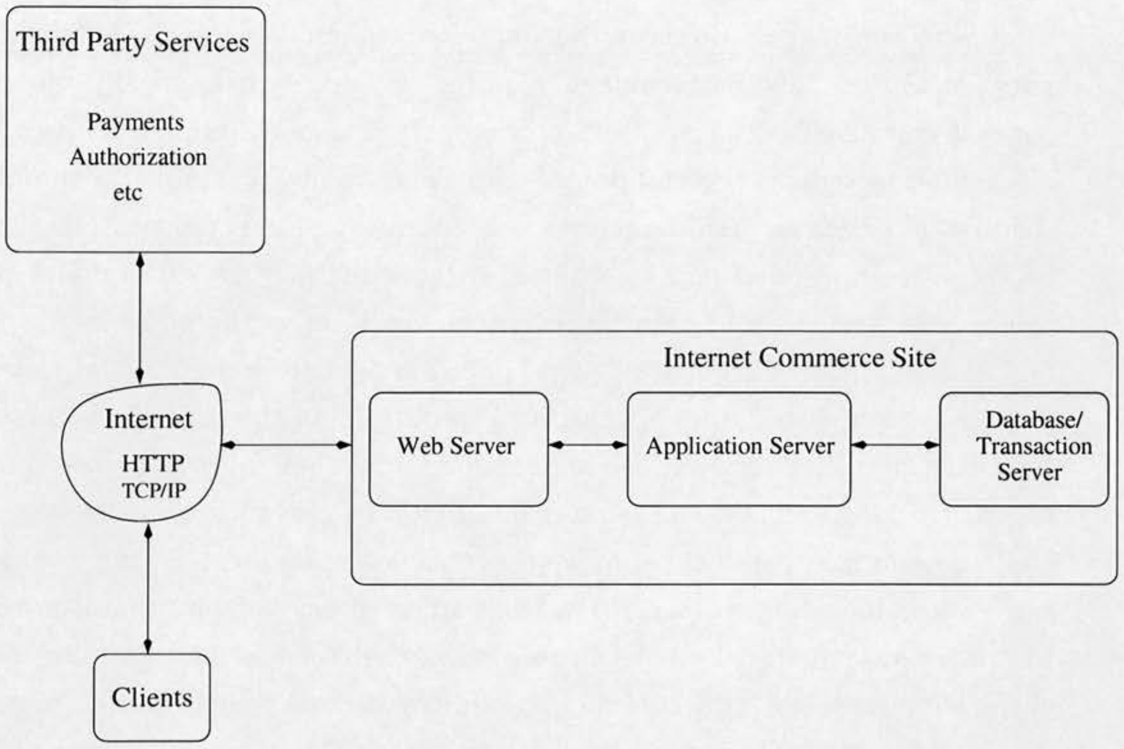


Figure 2.2: Internet Commerce System Architecture

Server, an Application Server and a Database Server. Third party services such as payment and certification are accessed via the Internet. The site's servers are composed of hardware and software components. Many hardware configurations are possible for hosting the system's servers ranging from hosting the three servers on one machine to hosting each of the servers on a cluster of machines. In the rest of this section, the software element of such servers is described.

The Web Server is the front-end of the system which interacts with customers through the HyperText Transfer Protocol (HTTP) implemented on top of TCP/IP. A Web Server listens for HTTP requests and follows the Request-Response paradigm where a response is returned for each request received. Multiple client requests can be handled by a Web Server simultaneously using process forking and multi-threading. Secure client connections can be established with Web Servers (especially in payment-related interactions) using security protocols such as SSL [18] and SET [19]. SSL authenticates the Web Server and provides a secure link over the Web on which private information can be passed. SET authenticates all parties involved in a transaction as well as providing secure links over the Web.

A Web Server responds to static page requests directly by returning the files composing such pages. However, dynamic page requests are forwarded to the Application Server. The business logic required to generate dynamically generated pages is contained in the Application Server. It acts as an interface between the manipulation and the presentation of data. The Application Server manipulates the system's data by issuing queries and updates to the Database/Transaction Server. Presentation of data is achieved by constructing pages containing query results. Different Application Server technologies are available including PHP and Java Servlets which are reviewed in Section 4.2 and Section 4.3 respectively. Due to the stateless nature of the HTTP protocol and the dependency between Internet Commerce services, the application server should provide some form of session tracking to maintain customer information across subsequent requests, including items selected in the shopping cart. Different session tracking techniques can be used including *cookies* [20] which is a named piece of data that is provided by the server and stored on the client's side. The *cookie* is added to subsequent client requests to the site. Another session tracking technique is *URL rewriting* in which the necessary session information is encoded into any URLs the server generates. Other forms of session tracking include using an application server technology's built-in session tracking system or to use persistent storage such as a database to store user session information.

A back-end data storage is required in an Internet Commerce system which is hosted by a Database/Transaction Server. It includes data such as the catalogue of products offered by the system, customer and order details. As described above, the Application Server issues queries and updates manipulating the data stored.

Transactions involving third-party independent institutions and companies are usually required in an Internet Commerce system. Payment authorisation is one form of such external transactions, e.g. customer's credit card details are submitted to the issuing bank (via the merchant's bank) for authorisation. Other forms of third-party requests include obtaining digital certificates required for authentication and making orders to the merchant's suppliers.

2.4.5 Internet Commerce Workload Characterisation

As described previously, workload characterisation is required for the performance modelling of Internet Commerce systems. In this section, the work done on the characterisation of web server workload is reviewed. This is followed by a discussion of characterisations specifically targeting Internet Commerce systems. Finally, benchmarks and tools that enable the generation of synthetic web and Internet Commerce workloads are reviewed.

Extensive work has been done on generating synthetic web server workload, with few studies specifically targeting Internet Commerce systems. Several properties (invariants) can be used to characterise web server workload. Based on the invariants identified by Arlitt and Williamson [21] [22], the following list highlights the main features that characterise a web server workload:

- File Types: the different file types composing the documents offered by a web site, e.g. HTML, dynamic content scripts, graphics, audio, video, etc.
- Median Document Size: for different documents offered by a web site.
- File Size Distribution: that is the cumulative size distribution of documents offered.
- Inter-reference time: this could be the time between references to the same document (as proposed by [21] [22]) but can be generalised to represent the user think time between requests to different documents in a web site.
- User Access Patterns: the services offered by an Internet Commerce systems are dependent; that is the user's decision of which document to request next is dependent on the outcome of the document request just completed. Thus, a set of possible next document requests must be identified for each document in a web site. A probability of selection is assigned to each next document based on observed user behaviour. Arlitt and Williamson [21] [22] only consider *Temporal Locality* for individual documents in terms of how frequently a document is requested in a short time interval.

Some of Arlitt and Williamson's [21] [22] findings when analysing access logs of six web sites include: *Temporal Locality* of different web site documents follows a non-uniform pattern where some documents are very popular and requests to them represent most of the workload and other documents are rarely requested. *Inter-reference times* are independent and follow an exponential distribution. *File Size Distribution* follows the Pareto distribution. They have also found that there is *self-similarity* in the workload of some of the sites they considered. That is, bursts can be observed at different time scales with no changes to the traffic structure. Crovella and Bestavros [23] confirm that WWW traffic has characteristics consistent with *self-similar* traffic. Their conclusion that *File Size Distribution* follows the Pareto distribution confirms Arlitt and Williamson's findings [21] [22]. Almeida *et al* [24] have investigated web documents frequency of reference and have found that it follows a Zipf-like distribution also confirming Arlitt and Williamson's conclusion.

Substantial work has been done on modelling web user's access patterns. Huberman *et al* [25] present the *law of surfing* in which a user continues to navigate a path in the web site until the expected cost of continuing is larger than the reward given by the expected value of the information to be found along this path. They use a random-walk model to present regularities in user access pattern and have found that the probability distribution of surfing depth follows a two-parameter inverse Gaussian distribution. Several predictive models have also been devised to predict the user's future requests. Schechter *et al* [26] use *point* and *path* profiles generated from the analysis of web server logs to predict user requests. A *point profile* for a web document includes the set of its selected next navigation options and the frequency with which each option was selected. A *path profile* contains a sequence of document requests made by a single user and the number of times that path occurred over the time period of the profile. Schechter *et al* propose predicting the next navigation by matching the user's current navigation sequence with the paths in the path profile. Markov models have also been used to predict web server user's navigation. Padmanabhan and Mogul [27], and Bestravos [28] use a first-order (memoryless) Markov model to predict the user's future navigations in order to improve web cache prefetching (as described in Section 5.2.2). To overcome the inaccuracy of first-order Markov models and the high state space requirements of higher order Markov modelling (when some history is taken into account), Pitkow and Pirolli [29] propose identifying longest repeating sequences of web user accesses and use those for navigation prediction. Another approach considered by Deshpande and Karypis [30] is to use selective Markov modelling in which high-order Markov models are optimised by eliminating states that are expected to have low prediction accuracy. Su *et al* [31] and Li *et al* [32] consider N-gram modelling commonly used in speech processing to predict user navigation.

Fewer studies have considered the specific characterisation of Internet Commerce workloads. To explore the properties of Internet Commerce workload, Arlitt *et al*

[33] analysed the logs of a large Internet Commerce system. They found the following: most of the requests handled by the system are for dynamically generated web pages. Popularity of different interactions follow a Zipf distribution which is consistent with the general web server workload properties described above. The average number of requests per session is about 7.5 for users and 51.8 for robots. Menasce *et al* [34] also analysed the workload of two Internet Commerce sites: an online bookstore and an electronic auction site. Some of their findings are: 88% of sessions are composed of less than 10 requests and most sessions last for less than 1000 sec. More than 70% of the functions requested are product selection functions rather than product purchasing functions. Strong self-similarity is found in the arriving traffic and the popularity of interactions follow a Zipf distribution. Menasce et al. also found that at least 16% of the requests are generated by robots. Almeida *et al* [35] provide a more detailed study on the presence of robots in Internet Commerce workload in the form of Crawlers and Shop-Bots. To characterise Internet Commerce workload access patterns, Menasce and Almeida [36] devised the Customer Behaviour Model Graphs (CBMGs). These are based on analysing HTTP logs where user sessions with similar behaviour are clustered into groups. The result of this analysis is to define CBMGs that describe the user behaviour pattern. Nodes of the graph represent pages on the site and arcs between states represent transitions between pages. Numbers are used along arcs to indicate the probability of making a transition. The use of clustering to understand Internet Commerce user behaviour is also considered by Wang *et al* [37] where different criteria of clustering are used, including user access patterns. Another characterisation of Internet Commerce workload is provided by the TPC-W benchmark [3] presented by the Transaction Processing Council (TPC). TPC-W provides a realistic model of an online bookstore site and uses the Emulated Browser (EB) entity to simulate the activities of concurrent web browsing e-commerce users, each autonomously traversing the bookstore web pages. A detailed description of TPC-W and its workload characterisation is provided in Section 3.2 and Section 5.5.1. Throughout this thesis, TPC-W characterisation of Internet Commerce workload is adopted.

Researchers have also considered predicting Internet Commerce user future navigation. For that, not only the history of users' traversal pattern is used, as in the general web server workload prediction described above, but also purchase patterns are considered to ensure an accurate prediction of user next requests [38] [39].

Several tools are available to generate general synthetic web server workloads. SPECWeb [40] and WebStone [41] are benchmarks that enable the simulation of a fixed number of web clients. Other tools such as s-client [42] and httperf [43] can sustain load with no restrictions on the number of users involved. A more realistic tool is SURGE [44] which can simulate more variable workloads that show *self-similarity*. A more specific e-commerce workload generation tool is Geist [45] which enables the generation of static web pages requests as well as requests for dynamic pages.

2.4.6 QoS in Internet Commerce

Different parties interacting with an Internet Commerce system have different views of its QoS and are directly affected by different aspects of the system's performance. Main stakeholders in an Internet Commerce system can be identified as: Customers, system administrators and business owners. The QoS metrics associated with each of them are described in this section.

From the customer perspective, three main QoS metrics can be identified which are: response time, service time and request failure rate. Response and service times are measures of the delay experienced by the customer when interacting with the system. The difference between them is that response time includes end-to-end delay covering network latency while service time only measures the delay caused by the system. Request failure rate is another QoS metric that affects the customer's view of the system. It represents the rate at which customer requests are not fulfilled successfully per unit time.

System resources utilisation is a main concern for Internet Commerce system administrators as they aim to ensure that there are no system bottlenecks degrading performance and that all system resources are optimally exploited. Examples of such resources include CPU and memory in different parts of the system. Throughput per unit time is another QoS metric usually monitored by an administrator as system performance is investigated. The request failure rate described above could be controlled by an administrator to reduce system load in a state of congestion by rejecting customer requests.

The Internet Commerce business owner's main measure of success is the system revenue. This financial aspect of Internet Commerce QoS can be measured by a profit per unit time metric. Certainly, such a metric is affected by the more technical measures of system QoS described above.

Non-functional QoS of Internet Commerce, such as system usability and security, are not considered in this thesis.

2.5 Summary

In this chapter, a general background on e-commerce systems was provided. Different types of e-commerce systems were reviewed and it was clarified that throughout this thesis, only Internet Commerce systems are considered, in particular, the retailing model of Internet Commerce. The elements required in the performance modelling of Internet Commerce systems were discussed including, the business model, the system's architecture, workload characterisation and the QoS metrics required to evaluate system performance.

In the next chapter, the design and implementation of ECBench, a general-purpose e-commerce performance benchmarking tool is described. ECBench has been developed to aid in the making of QoS-oriented decisions at early stages of the system design process.

Chapter 3

ECBench: Design and Implementation

3.1 Introduction

In many e-commerce systems, timely behaviour is crucial in order to maintain the site owner's competitive edge: poor performance can quite literally translate into lost revenue [10]. However, little systematic work has been done on analysing the performance of such systems and their supporting technologies. In this chapter, a tool, ECBench, is described which aims to provide a framework in which different e-commerce technologies and techniques can be easily benchmarked against the standard TPC-W benchmark [3]. In contrast with other commercial tools available such as e-TEST [46], ECBench is specialised in Internet Commerce, rather than general web applications. Its modular design ensures that it provides a general-purpose Internet Commerce performance benchmarking framework.

The TPC-W benchmark has been developed by the Transaction Processing Performance Council (TPC) as a response to the rise of e-commerce systems. It specifies the behaviour of an online bookstore, including many of the elements commonly found in e-commerce applications: a web-site supported by a web serving component which can present both static and dynamic web pages; and a relational database which is accessed from the web server to provide transaction processing and decision support. Moreover the benchmark also specifies emulated remote browsers for different classes of users, providing the workload on the system.

ECBench can aid an Internet Commerce application developer to choose between alternative technologies based on their performance merits to achieve better overall performance. ECBench can also be used by a researcher to investigate and analyse new techniques to improve e-commerce system performance.

ECBench is a flexible and extendable tool due to its modular design. Basing the tool on the TPC-W specification ensures that it provides a realistic model of an Internet Commerce system. The tool incorporates support for experiment design, allowing the user to replicate runs of the benchmark and compare the results obtained with different workload mixes.

Two papers [1], [2] on ECBench have been accepted and presented; the first at the 18th UK Performance Engineering Workshop 2002 (UKPEW2002), in which the design and implementation of ECBench are described. This chapter is an extended version of this paper. The second paper was presented at the 10th International Conference on Analytical and Stochastic Modelling Techniques and Applications 2003 (ASMTA2003). It includes a case study on the use of ECBench to benchmark PHP and Java Servlets. A detailed description of the case study is provided in Chapter 4

The rest of this chapter is structured as follows. In Section 3.2, the TPC-W specification is reviewed. This is followed in Section 3.3 by a description of ECBench's design objectives. An overview of the tool design is given in Section 3.4. This is followed in Section 3.5 by a description of the tool's main control component. The workload part of ECBench is then explained in Section 3.6. ECBench's Web Server and Database Server are described in Section 3.7 and Section 3.8 respectively. In Section 3.9, implementation issues are discussed. Finally, in Section 3.10, a chapter summary is provided.

3.2 The TPC-W Benchmark

The TPC-W benchmark has been developed by the Transaction Processing Performance Council (TPC) [3], a consortium of system and database vendors. Historically, TPC has specified standard benchmarks (e.g. TPC-C, TPC-H) for evaluating the performance of both transaction processing and decision support database systems. One of its latest benchmarks is TPC-W, an e-commerce-specific benchmark. TPC-W Version 1 specifies the behaviour of an on-line bookstore, including the three main components of an e-commerce application: remote browsing, web server and database server. TPC-W Version 2, due to be published

at the end of year 2004, focuses on B2B e-commerce systems. For the remainder of this thesis, "*TPC-W*" should be treated as synonymous with "*TPC-W, Version 1*", unless otherwise stated.
TPC-W's remote browsing specifications are outlined next in Section 3.2.1. This is followed in Section 3.2.2 by an overview of TPC-W's web server component. TPC-W's database server specifications are then summarised in Section 3.2.3. Finally in Section 3.2.4, the measurement intervals specified by TPC-W are outlined.

3.2.1 Remote Browser Emulator

The workload component of ECBench is based on TPC-W's Remote Browser Emulator (RBE), which is a specification for a set of Emulated Browsers (EBs). EBs simulate the activities of concurrent web browsing e-commerce users, each autonomously traversing bookstore web pages and making requests to a web server. Each EB can represent one of three classes of users: a customer, a new user or a site administrator.
As described in Section 3.2.2, TPC-W defines 14 web interactions which can be requested by an EB. During its lifetime, an EB requests a sequence of these web interactions moving from one interaction to the next, in the same way that a web browsing user navigates a site, clicking one hypertext link after another. TPC-W specifies the next possible navigation options that can be requested by an EB on completion of each of the web interactions defined in the benchmark. Threshold integer values between 1 and 9999 are specified for each navigation option (one for each measurement interval, see Section 3.2.4). These threshold values can be considered as defining probabilities governing the navigational pattern for a user in the TPC-W model. Appendix C includes the navigational threshold values for a TPC-W's ordering interval. To select its next request, an EB generates a random number, from a uniform distribution between 1 and 9999. It then selects the navigation option for which the threshold is equal to or most immediately greater than the random number. The EB spends a random period of time (*Think Time*) sleeping between subsequent web interactions. This emulates the user's think time and is generated from an exponential distribution.
User-specific information must be maintained in an EB, including session tracking details and customer identification.

3.2.2 Web Server

The TPC-W benchmark defines 14 web interactions to be supported by a web server component which are: Home, Shopping Cart, Customer Registration, Buy Request, Buy Confirm, Order Inquiry, Order Display, Search Request, Search Result, New Product, Best Sellers, Product Detail, Admin Request and Admin Confirm. The interactions vary in the amount of server-side processing they need. Some require dynamically generated HTML pages and one or more database operations. Others are relatively lightweight, requiring only web serving of static HTML pages and images. For each web interaction, TPC-W specifies its input requirements, processing definition, response page definition and EB navigation options which are the set of web interactions that can be selected by the EB on completion of the interaction.
As described in Section 2.4.4, session tracking is vital to any e-commerce application in order to retain information such as shopping carts and customer IDs from one HTTP request to another. TPC-W suggests two techniques for session tracking which are URL-rewriting and cookies.

3.2.3 Database Server

The TPC-W benchmark defines the exact schema used for an on-line bookstore database. The schema consists of eight tables: customer, address, order, order line, credit card transaction, item, author and country. Additional tables may be added to the schema. The size of the database depends on two factors: the number of EBs that will be used as a workload and a *scale factor*, an integer with a possible value of 1000; 10,000; 100,000; 1,000,000 or 10,000,000.

3.2.4 Measurement Intervals

Three distinct measurement intervals are specified by TPC-W: shopping interval, browsing interval and ordering interval. They are distinguished by the ratio of browsing-related web pages visited to ordering-related web pages visited during the measurement interval. The shopping interval is intended to reflect a shopping scenario, in which 80% of the pages the user visits are related to browsing and 20% are related to ordering. In a browsing interval, ordering pages visited go down to 5% whereas in an ordering interval the ratio of browsing and ordering is even.

3.3 ECBench Design Objectives

A set of design criteria were set, and met, in the design of ECBench. These include the following:

- Realistic e-commerce modelling was an important design criterion. This was ensured in the design by adopting TPC-W as a standard e-commerce specification. Not only are different e-commerce site components represented faithfully; the design also includes realistic workload generation capabilities based on TPC-W's e-commerce access patterns.

- Developing a flexible, extendable tool was a major concern. This led to a modular design in which incorporating a new technology/technique for benchmarking just involves adding a new module to the tool.

- ECBench provides a set of performance metrics necessary for assisting e-commerce technology/technique performance evaluators. Metrics such as response time, frequency distributions of different web interactions, throughput, profit per second, CPU utilisation and memory utilisation can be analysed and presented graphically.

- The tool provides experiment design features allowing the user to specify a number of experiments. For each experiment, factor level combinations such as size of workload driving the system, size of store and measurement interval structure can be specified. The number of replications for each experiment can also be specified.

- Ensuring that the analysis and presentation of experimental results have minimal overhead was an objective from the early stages in the design, thus preserving the practical usefulness of the tool. This resulted in a local data collection strategy in the design. Experimental data results are maintained locally by different parts of the system. Results are gathered at the end of each experiment from different parts of the system for analysis and presentation.

ECBench is novel in its provision of a performance benchmarking framework specialised in Internet Commerce rather than general web applications. This is ensured by basing the design of ECBench on the TPC-W standard specification and supporting a comprehensive QoS metric set important to the performance analysis of e-commerce applications. Experiment design, execution and result analysis is facilitated by ECBench's experimentation features. The design of

ECBench as a general-purpose Internet Commerce tool is the other aspect of its novelty. The modular design of ECBench makes it easily extendable to benchmark new e-commerce technologies and techniques.

3.4 ECBench Design Overview

A block diagram illustrating ECBench design is shown in Figure 3.1. The tool is composed of three main components: the server-side of an e-commerce system, i.e. the System Under Test (SUT), a Remote Browser Emulator workload component (RBE) and a component controlling different parts of the tool (Control Unit). The System Under Test (SUT) emulates the server side of a typical Internet Commerce system consisting of a Web Server (which includes an Application Server) and a Database Server. RBE is the component responsible for generating and maintaining the workload on the SUT by emulating customers navigating an e-commerce site. The final main component in ECBench is the Control Unit which provides features for experimental design, data gathering and analysis. It is also responsible for controlling the setup and maintenance of experiments on various parts of the tool.

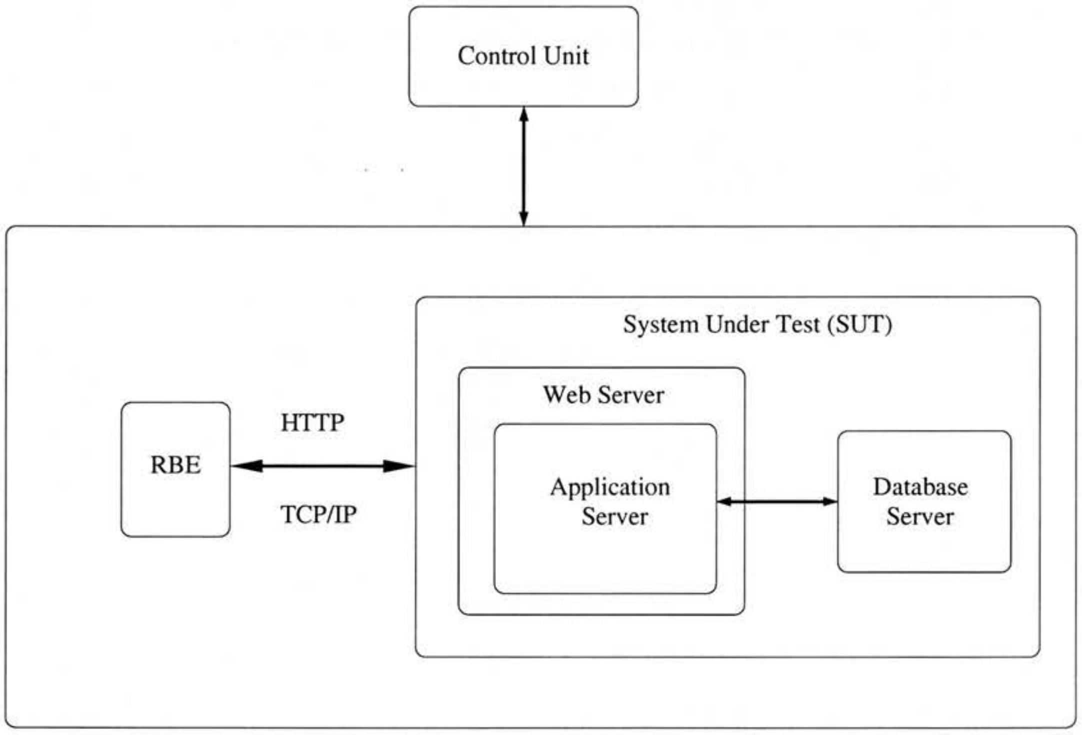


Figure 3.1: ECBench Overall Design

ECBench's Control Unit is described next in Section 3.5. This is followed in Section 3.6 by a description of ECBench's workload part, the RBE. The Web Server of the SUT, including the Application Server model used, is described in Section 3.7. Finally, in Section 3.8 a description of the Database Server part of the SUT is given.

3.5 Tool Control

The Control Unit is the central component of ECBench. In addition to allowing the user to specify experiment setup via a GUI, the Control Unit is also responsible for setting up and maintaining the running of experiments, collecting and analysing experiment results. It contains a controlling subcomponent for each part of the tool. Thus, controlling a new extension to ECBench would require only adding a new controlling subcomponent to the Control Unit, meeting the flexiblity criterion described in Section 3.3.

Sub-components composing the Control Unit are illustrated in Figure 3.2. The Experimental Design entity is used to store user experiment design details passed via the GUI component. The Control Unit sends instructions to the controllers of different parts of the benchmark including the RBE Controller, Web Server Controller and Database Server Controller for the setup and maintenance of experiments. After finishing an experiment run, the Control Unit requests the Result Analysis

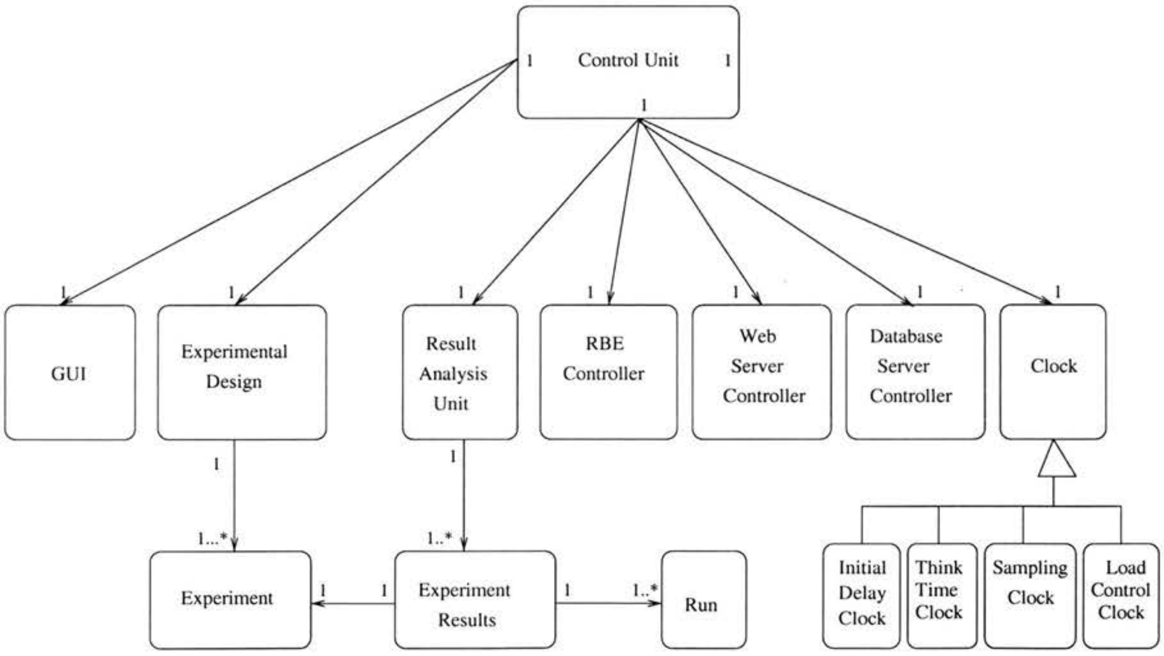
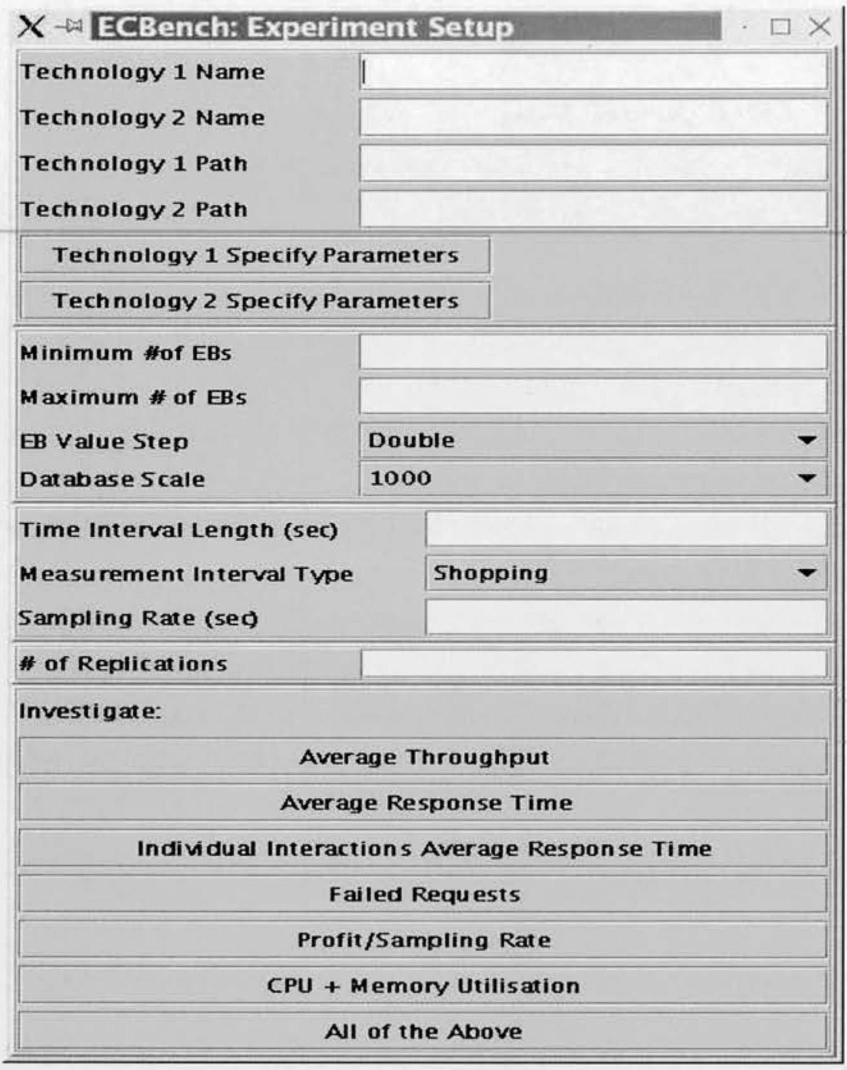


Figure 3.2: Control Unit

Unit to collect data results from different parts of the system for analysis. The Clock and its subcomponents are responsible for controlling time in the tool.
In the following subsections, we describe each of these subcomponents in more detail.


ECBench: Experiment Setup

Technology 1 Name
Technology 2 Name
Technology 1 Path
Technology 2 Path
Technology 1 Specify Parameters
Technology 2 Specify Parameters
Minimum #of EBs
Maximum # of EBs
EB Value Step: Double
Database Scale: 1000
Time Interval Length (sec)
Measurement Interval Type: Shopping
Sampling Rate (sec)
# of Replications
Investigate:
Average Throughput
Average Response Time
Individual Interactions Average Response Time
Failed Requests
Profit/Sampling Rate
CPU + Memory Utilisation
All of the Above

Figure 3.3: ECBench Experiment Setup GUI

3.5.1 Experiment Design

ECbench provides experimental design capabilities for the user to be able to specify values of different parameters in the simulation using the GUI component shown in Figure 3.3. The Experimental Design component is responsible for holding the experimental plan of the user. As the user is allowed to specify a number of experiments which are run sequentially, Experimental Design consists of a set of Experiment objects (see Figure 3.2), each holding the design details for one experiment. The following parameters can be defined in an Experiment object:

- Technologies/techniques benchmarked, up to two e-commerce technologies/techniques can be specified to be benchmarked in any experiment. Implementation locations of such technologies/techniques are required.
- Specific parameters, a maximum of 10 specific parameters could be specified for each technology/technique benchmarked.
- Number of replications, which is the number of times an experiment run is repeated.
- Workload details including workload size which is the number of Emulated Browsers (EBs) emulating customers generating requests to the SUT (as described in Section 3.6). A lower and upper limit on the size of the workload can be specified. The size could be varied across experiment runs or gradually for each run (by a step size value specified by the user).
- Measurement interval which can be a shopping, a browsing or an ordering interval (see Section 3.2.4). The time period of the interval must be specified.
- *Ramp up* period specifying the time period that must be elapsed before measurements are recorded. This delay is required to ensure that the workload has reached steady state before measurements are taken.
- *Size of store* which is a scale factor (with possible values of 1,000; 10,000; 100,000; 1,000,000; 10,000,000) contributing to the determination of the bookstore database size (see Section 3.2.3).
- Performance metrics measured. A number of performance metrics could be specified to be measured including average throughput, average response time, individual interactions' average response time, failed requests, profit per second, server CPU utilisation and server memory utilisation.
- Sampling rate: that is, the frequency at which measurements are taken in an experiment run.

The Experimental Design component informs the Control Unit of the experiment design details needed at each stage of the simulation. This would include experiment parameters for the experiment run to be performed next. The Control

Unit uses these parameters to configure different parts of the tool through their respective controllers.

3.5.2 Experiment Results Analysis

ECBench provides analysis features for results obtained from performing an experiment composed of a set of experiment runs. The Result Analysis Unit (see Figure 3.2) consists of a set of Experiment Results objects holding the results for each experiment performed. Each Experiment Results object is associated with an Experiment object and contains a set of Run objects holding the results of each experiment run performed.
During an experiment run, experimental data is recorded locally by different parts of the system minimising data recording overhead (cf. the minimum overhead criterion in Section 3.3). On completion of the run, the Result Analysis Unit receives a request from the Control Unit to gather the experimental data which are then stored in a Run object. Thus on completion of all runs for an experiment, the Result Analysis Unit will have a collection of Run objects holding the results obtained from each run.
Data analysis is done after the completion of each experiment. Graphs are then produced for each performance metric required.

3.5.3 Time Management

ECBench provides a dynamic continuous simulation where the state of the simulation changes continuously with time. The Clock subcomponent of the Control Unit (see Figure 3.2) is responsible for managing time in ECBench. It is the central timing component in the tool which ensures the correct duration of experiment runs and distributes timing information to all parts of ECBench.

Other clocks are used to manage specific time-related activities in an ECBench simulation. They include the Initial Delay Clock, Think Time Clock, Sampling Clock and Load Control Clock and these inherit from the main Clock component. The use of each specific clock is described below:

- Initial Delay Clock: used to ensure the *ramp up* period described in Section 3.5.1. The recording of measurements in an experiment run will start only after the Initial Delay Clock delay elapses.
- Think Time Clock: the think time an e-commerce user spends making a decision on the next navigation option to be taken is emulated as described

in Section 3.6.1. Think Time Clock ensures that the next navigation option is not taken until the think time elapses. The period of the think time is determined randomly by an exponential distribution.

- Sampling Clock: this clock manages the frequency at which measurements are taken in an experiment run. Each time a sampling interval (specified in the experiment design) elapses, the Sampling Clock triggers different parts of ECBench to take measurements. It was decided to use fixed-interval sampling rather than event-based to reduce the volume of samples taken in order to minimise overhead.

- Load Control Clock: this clock manages the workload of the simulation. It triggers the change in the workload size either toggling between low and high load or gradually increasing the workload as specified in the experiment design.

3.5.4 Workload Control

The workload part of ECBench is controlled by the Control Unit's RBE Controller subcomponent shown in Figure 3.2. RBE Controller interacts with the RBE (through the Browsing Control unit described in Section 3.6) sending it configuration and control information from the Control Unit and receiving experiment measurements collected by the RBE. It informs the RBE of relevant experiment design details including:

- Implementation locations of the current technology/technique being benchmarked.

- Any specific parameters that are related to the benchmarked technology/technique.

- The size of the workload (in terms of the number of EBs) required at each stage of the simulation.

- The *ramp up* period during which EBs should be created.

- Measurement-related details including the type of the current measurement interval, workload-related performance metrics for which measurements are required and the sampling rate at which measurements should be taken.

The RBE Controller also passes control signals to the RBE including the start/finish of an experiment run and sampling signals. On completion of each experiment run, measurements collected by the RBE are passed to the Control Unit through the RBE Controller.

3.5.5 Web Server Control

As described in Section 3.7, the Web Server part of ECBench represents the first part of the SUT. The Control Unit configures and manages the Web Server and its Application Server using the Web Server Controller subcomponent (see Figure 3.2). The Web Sever Controller communicates with the Web Server sending it experiment design details and control signals and receiving experiment measurement collected by the Web Server.
The following experiment design details are passed to the Web Server in the course of an experiment:

- The e-commerce technology/technique to be used in the experiment and the location of its implementation.
- Parameters which are related to the Web Server operations and specific to the technology/technique being benchmarked.
- Measurement-related details including the performance metrics for which measurement is required, the starting time of the measurement period, the length of the measurement interval and the sampling rate.

On completion of each experiment run, measurements collected by the Web Server are passed to the Control Unit through the Web Server Controller.

3.5.6 Database Control

The Database Server of ECBench represents the second part of SUT (see Section 3.7). It is controlled by the Control Unit via its Database Server Controller subcomponent shown in Figure 3.2). It informs the Database Server of experiment design details including:

- Information required to populate the database including the size of the workload (number of EBs) and the *scale factor*.
- Specific parameters to the technology/technique being benchmarked which are related to the Database Server operations.
- Measurement-related details including the performance metrics for which measurement is required, the starting time of the measurement period, the length of the measurement interval and the sampling rate.

On completion of each experiment run, measurements collected by the Database Server are passed to the Control Unit through the Database Server Controller.

3.6 User Browsing Emulation

ECBench provides an Internet Commerce workload based on the TPC-W standard described in Section 3.2.1. The RBE of ECBench is responsible for generating and driving the tool's workload. As shown in Figure 3.4, it includes a set of Emulated Browsers (EBs) emulating web browsing e-commerce users requesting web interactions from a web server. The RBE also contains a Browsing Control entity responsible for managing the creation and maintenance of EBs according to instructions received from the Control Unit (see Section 3.5.4). The EB sub-

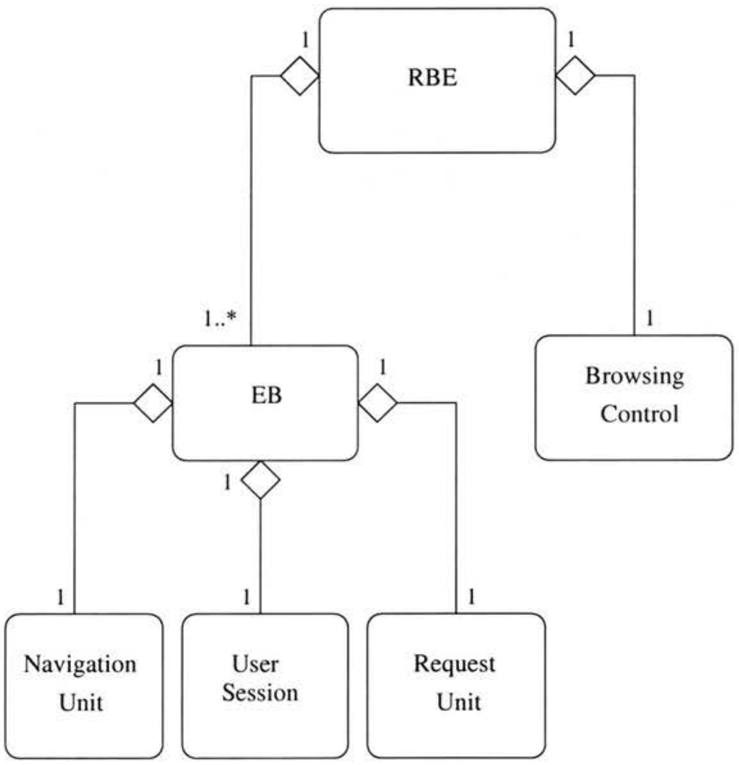


Figure 3.4: User Browsing Emulation

component is designed to emulate a web browser used by an e-commerce user who can be a *New User* or a *Customer*. The *Administrator* user type is not supported in ECBench workload as the number of administrator-related requests can be neglected if compared to the number of requests generated by *New User* and *Customer* users. As shown in Figure 3.4, an EB consists of three subcomponents, a Navigation Unit, a Request Unit and a User Session. Its Navigation Unit emulates the user's navigational behaviour in an on-line bookstore. The EB's User Session sub-module is designed to emulate the maintenance of e-commerce user-specific information in a web browser including session tracking details and customer identification. The sending and receiving of HTML content via HTTP and TCP/IP over a network connection is emulated by the Request Unit.

EB's navigation is described next in Section 3.6.1. The way user session information is maintained in an EB is discussed in Section 3.6.2. EB's emulation of the sending and receiving of HTML content via a network connection is described in Section 3.6.3. Finally, the control and maintenance of EBs is reviewed in Section 3.6.4.

3.6.1 Navigation

The navigational model used in ECBench is based on the TPC-W standard's user access patterns described in Section 3.2.1. Figure 3.5 illustrates the user navigation part of an EB. The Navigation Selector is the central component to the EB's user navigation. It is composed of a set of Web Interaction objects, a Threshold Generator and an instance of Think Time Clock which was described in Section 3.5.3.
A web service that can be requested by the user is represented by the Web Interaction component. Specific services inherit from Web Interaction including: Home, Best Sellers, New Products, Search Request, Search Result, Product Detail, Shopping Cart, Customer Registration, Buy Request, Buy Confirm, Order Inquiry and Order Display. Each Web Interaction contains a list of Navigation Option objects representing all possible next interactions that the user can request on completion of the Web Interaction. Navigation Option maintains a threshold value (specified by TPC-W) for each possible TPC-W measurement interval (browsing, shopping or ordering as described in Section 3.2.4) required to determine the next web interaction selected.

Navigation Selector contains one object from each of the Web Interaction's subclasses. It also contains a Web Interaction object representing the current web interaction being requested. Navigation Selector decides on the next interaction to be requested using the algorithm shown in Figure 3.6. The algorithm is based on the TPC-W threshold approach outlined in Section 3.2.1. At the start of the navigation, the first selection is always the Home interaction. On completion of each current interaction i, the Threshold Generator of the Navigation Selector (see

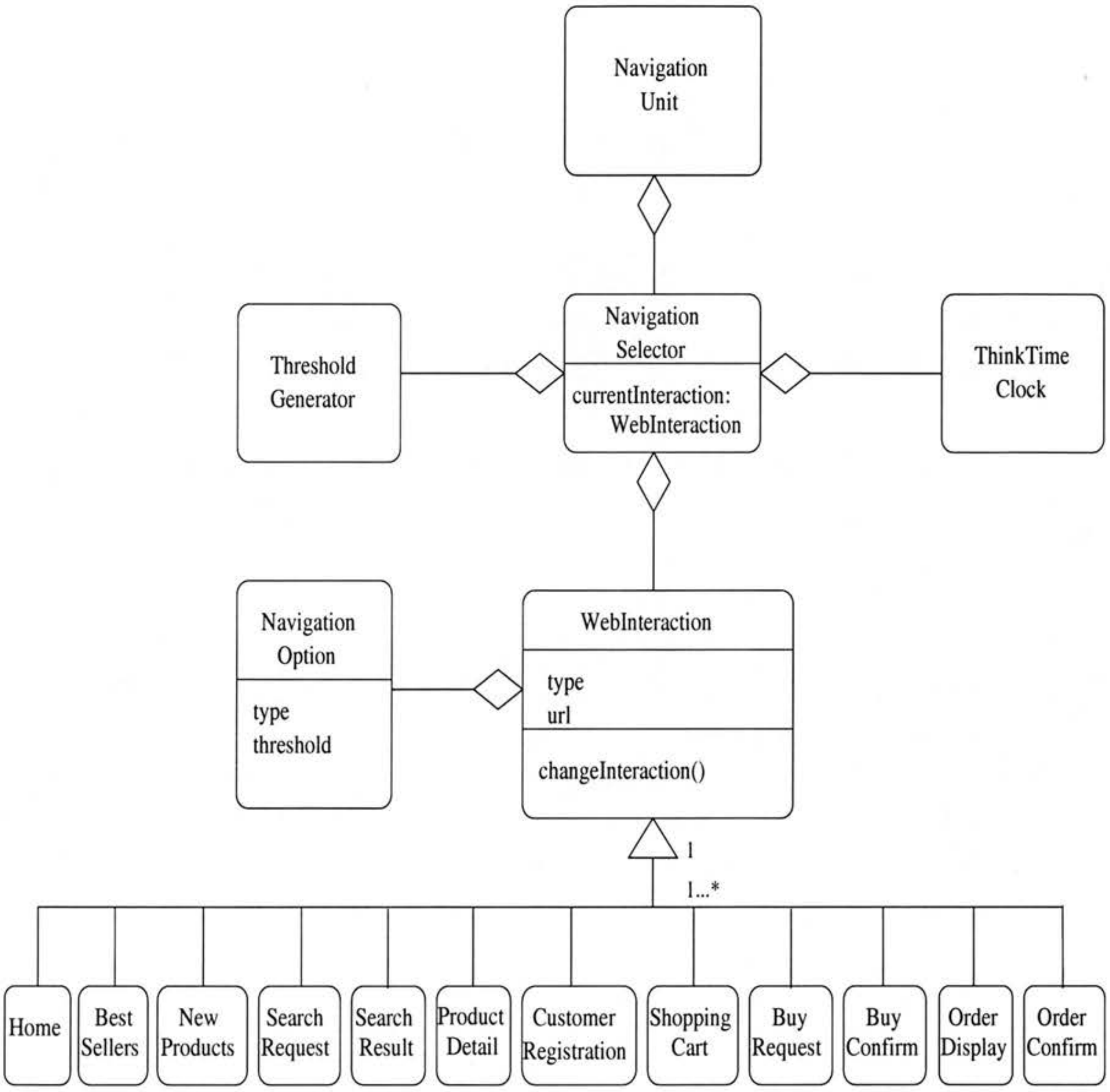


Figure 3.5: ECBench's User Navigational Part

Figure 3.5) generates a random number Threshold value T from a uniform distribution between 1 and 9999. T is subtracted from the current measurement interval's threshold value for each Navigation Option in the list of next navigation options for i. The Navigation Option with threshold equal or immediately greater that T is selected as the next navigation option to be requested.

Part of user behaviour when navigating a web site is a think time delay between web interaction requests. Formally, think time, TT, can be defined as follows:

$$TT = T1 - T2$$

```
Given that *currentInteraction* of Navigation Selector is $i$ and the next
Navigation Options for $i$ are $n_{i0}, n_{i1}, \ldots, n_{ij}$ with thresholds $t_{i0}, t_{i1}, \ldots, t_{ij}$
respectively:

if(first interaction)
    select Home
else
    generate a random Threshold value $T$ between 1 and 9999
    find $t_{ij}$ such that $t_{ij} - T$ is minimum and $t_{ij} - T \geq 0$
    select $n_{ij}$
```

Figure 3.6: Navigation Selection Algorithm (implemented in Web Interaction's *changeInteraction()* method)

where $T1$ is the time measured after the last byte of the last web interaction is received and $T2$ is the time measured before the first byte of the next interaction request is sent.

The Think Time Clock shown in Figure 3.5 ensures that a time delay representing the user think time elapses before the next interaction selected by Navigation Selector is requested. As described in the time control part of ECBench (see Section 3.5.3), an exponential distribution is used to randomly generate the think time between interactions. The processing time required to select the next interaction is subtracted from the the think time value generated.

On completion of the think time delay, the type of the next interaction selected is passed to EB's Request Unit to form and deliver a request for the interaction and to handle the data returned (as described in Section 3.6.3).

3.6.2 User Session Tracking

As described in Section 2.4.4, session tracking is required in e-commerce systems to retain customer information across different HTTP requests made in a user session. In ECBench, it was decided to use cookies to enable session tracking. The popularity of cookies compared with URL-rewriting is the main reason behind this choice. The maintenance of cookies in web browsers is modelled in ECBench by including a User Session sub-component in EB as illustrated in Figure 3.4.

The User Session is responsible for maintaining user-specific information which is added to EB requests. The main piece of information held in User Session is the Customer Id (C_Id) which is generated and maintained in the following way:

- If the type of the user is *New User*, there is no customer Id required for the first interaction. The Application Server randomly generates a C_Id using a non-uniform random function when the Customer Registration interaction is requested (as described in Section 3.7.1). C_Id is then sent back to the EB and stored in its User Session sub-component. In subsequent interactions, C_Id is added to all requests made.

- If the user type is *Customer*, a customer Id is required for the first interaction. A non-uniform function is used to generate a value for C_Id which is sent as a parameter to the first interaction (Home interaction described in Section 3.7.1) and stored in User Session.

Following the TPC-W specification, the User Session also contains timing information to control the expiry of the User Session including:

- User Session Minimum Duration (USMD), which is the minimum duration for which the User Session must last. USMD is generated randomly using a negative exponential distribution.

- User Session Current Duration (USCD), which is the elapsed time from the time just before the first request by the EB to the current time during the User Session.

The User Session is terminated when the all of the following conditions are true:

- Think Time has just completed following the completion of a web interaction,

- The navigation option chosen by the Navigation Selector for the next web interaction is the Home web interaction,

- and USCD >= USMD.

3.6.3 Request Unit

Part of web browser functionality is to form HTTP requests for web interactions selected by the user. The web browser then establishes HTTP/TCP/IP connections with a web server to fulfil such requests. The HTML documents returned,

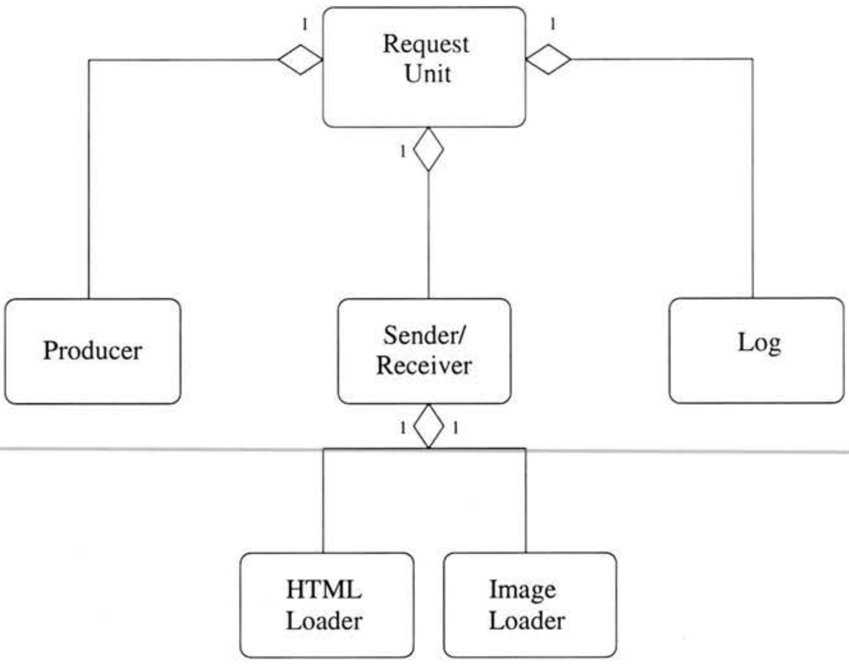


Figure 3.7: Request Unit Block Diagram

which often include images, are displayed on the browser.
These operations are modelled in ECBench by the EB's Request Unit component. Figure 3.7 illustrates the different parts composing the Request Unit. As described in Section 3.6.1, the Navigation Unit decides on the next web interaction to be requested. The decision is then passed to the Request Unit where its Producer subcomponent forms a request for the web interaction selected. A request is composed of the following:

- URL: Every web interaction has a URL which is specified by the user at the experiment design phase as described in Section 3.5.1. It specifies the location of the application server implementation of the current experiment.
- Input parameters and their values: Input parameters are required for each Web Interaction supported by ECBench's Application Server as described in Section 3.7.1. The values assigned to input parameters are generated randomly according to the TPC-W specification.
- Request type: GET or POST.
- User session information: If required, user information maintained by the User Session component of EB (see Section 3.6.2) is included in the request.

The Producer passes the request details to the Sender/Receiver sub-component for the request to be made. The Sender/Receiver packages the request into HTTP form. The HTTP/TCP/IP connection is then established for the request with the Web Server part of ECBench (see Section 3.7). The connection is associated with the User Session component of the requesting EB. The HTML document returned is handled by the TextHandler and the ImageHandler sub-components shown in Figure 3.7. TextHandler models the displaying of text on an internet browser by printing to standard output. Images are handled by ImageHandler with the number of images determined by the type of the web interaction.

The Request Unit's Log sub-component is responsible for gathering statistics on the web interactions handled by the Request Unit. When instructed by the Control Unit of ECBench (see Section 3.5.4), Log stores the following:

- Response time measurements of web interactions completed successfully grouped by web interaction type;
- Throughput of web interactions completed successfully grouped by web interaction type;
- Number of failed web interactions.

Log stores data locally which is collected by the Control Unit (see Section 3.5.1) at the end of the experiment, thus minimising overhead.

3.6.4 Browsing Control

In an ECBench experiment, the RBE contains a set of EBs representing the tool's workload. The control of such workload is the responsibility of the RBE's Browsing Control shown in Figure 3.4. Browsing Control is an interface to ECBench's Control Unit (see Section 3.5.4) through which instructions are sent to the RBE, via the RBE Controller described in Section 3.5.4.
The Browsing Control creates the required number of EBs gradually during the *ramp up* period. When creating an EB, Browsing Control specifies the type of the user represented by the EB: a *New User* or a *Customer*. The ratio of *Customers* to *New Users* is maintained at 4:1 by the Browsing Control. Browsing Control also informs the EB created of the type of the current measurement interval to determine the navigational pattern of the EB. Instructions on the rate at which measurements should be taken are also given.
During the operation of the EB, the Browsing Control sends *start* and *stop* signals to the EB instructing it to start navigation or to stop navigation when a reduction in the workload is required or if the measurement interval time has finished. A

resume signal can also be sent to the EB to instruct it to restart navigation after being suspended.
At the end of each experiment, the Browsing Control collects the data gathered by each of the EBs making the workload to be passed to the Control Unit for analysis.

3.7 Web Server

ECBench's web server models the web server component of an on-line bookstore which interacts with a database server (emulated by the Database Server described in Section 3.8). Figure 3.8 is a block diagram of ECBench's Web Server. Its Application Server handles EB HTTP requests for different web interactions supported by the tool. The Web Server is controlled by the Web Server Control sub-component shown in Figure 3.8.

The Application Server is described in Section 3.7.1 including the business logic associated with each web interaction supported. This is followed in Section 3.7.2 by a description of the control of the Web Server.

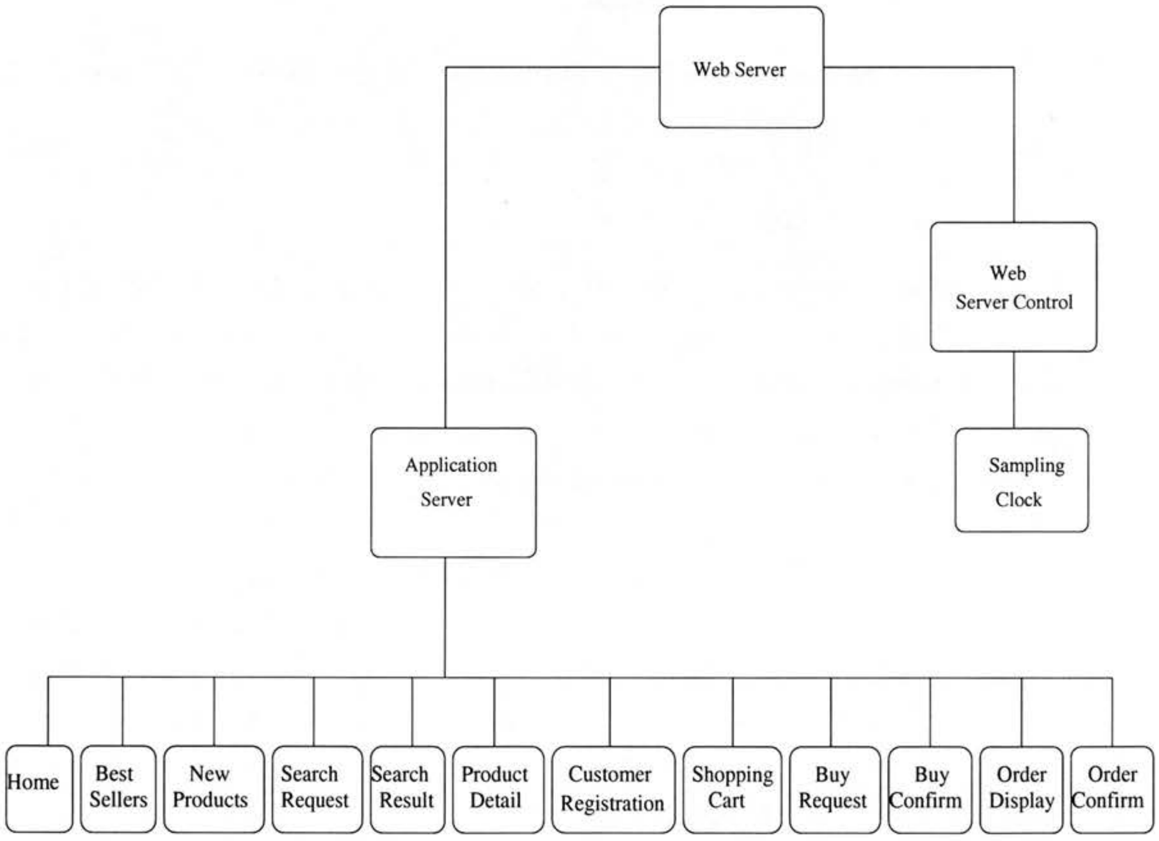


Figure 3.8: ECBench's Web Server Component

3.7.1 Application Server

The Application Server includes the business logic required to provide support for different web interactions in the bookstore e-commerce system defined in the TPC-W specification. Support is provided for the 12 web interactions (excluding administrator-related interactions) composing the Application Server, which are illustrated in Figure 3.8. They are of two types: *dynamic* where the content of the returned HTML page is generated on-the-fly using data obtained from database queries and *static* interactions where the content of the HTML document returned is fixed and the same HTML page is returned for all such requests. The HTML document returned by each web interaction includes images which represent logos, buttons for next navigation options and promotional thumbnails. The number of images returned by each web interaction are listed in Table 3.1.

Web Interaction	*Number of Images*
Home	9
BestSellers	9
New Products	9
Search Request	5
Search Result	9
Product Detail	6
Shopping Cart	9
Customer Registration	4
Buy Request	3
Buy Confirm	2
Order Inquiry	3
Order Display	2

Table 3.1: Number of Images Fetched for each Web Interaction

It was decided to use a standard image file of size 7kb, an average size for buttons and thumbnail images commonly displayed in on-line store web pages. Each web interaction has a list of next interactions that can be requested after its completion. Figure 3.9 illustrates the navigational sturcture of the server. Each web interaction has a URL address, could require input parameters and could involve database operations if it is a *dynamic* interaction. To keep the Application Server general, no specific security or payment protocols are provided in any of the server's interactions. The Application Server can be easily extended by an ECBench user interested in benchmarking such protocols. The following is a summary of each web interaction supported by the Application Server:

- Home: This is the first web interaction that is requested by an EB in a session. If the EB user requesting Home is a *Customer*, a customer id (C_Id) needs to be generated and passed by the EB with the Home request. In this case, a database transaction is executed to get customer details. The HTML code is then produced and returned to the EB.

- Best Sellers: A list of best seller items in a specific subject is returned to the EB requesting this interaction. A *subject* input parameter is required by this interaction which is used to make a database query to search for the 50 most sold items on that subject. The HTML document returned contains a list of the items obtained with links to the Product Detail page of each item.

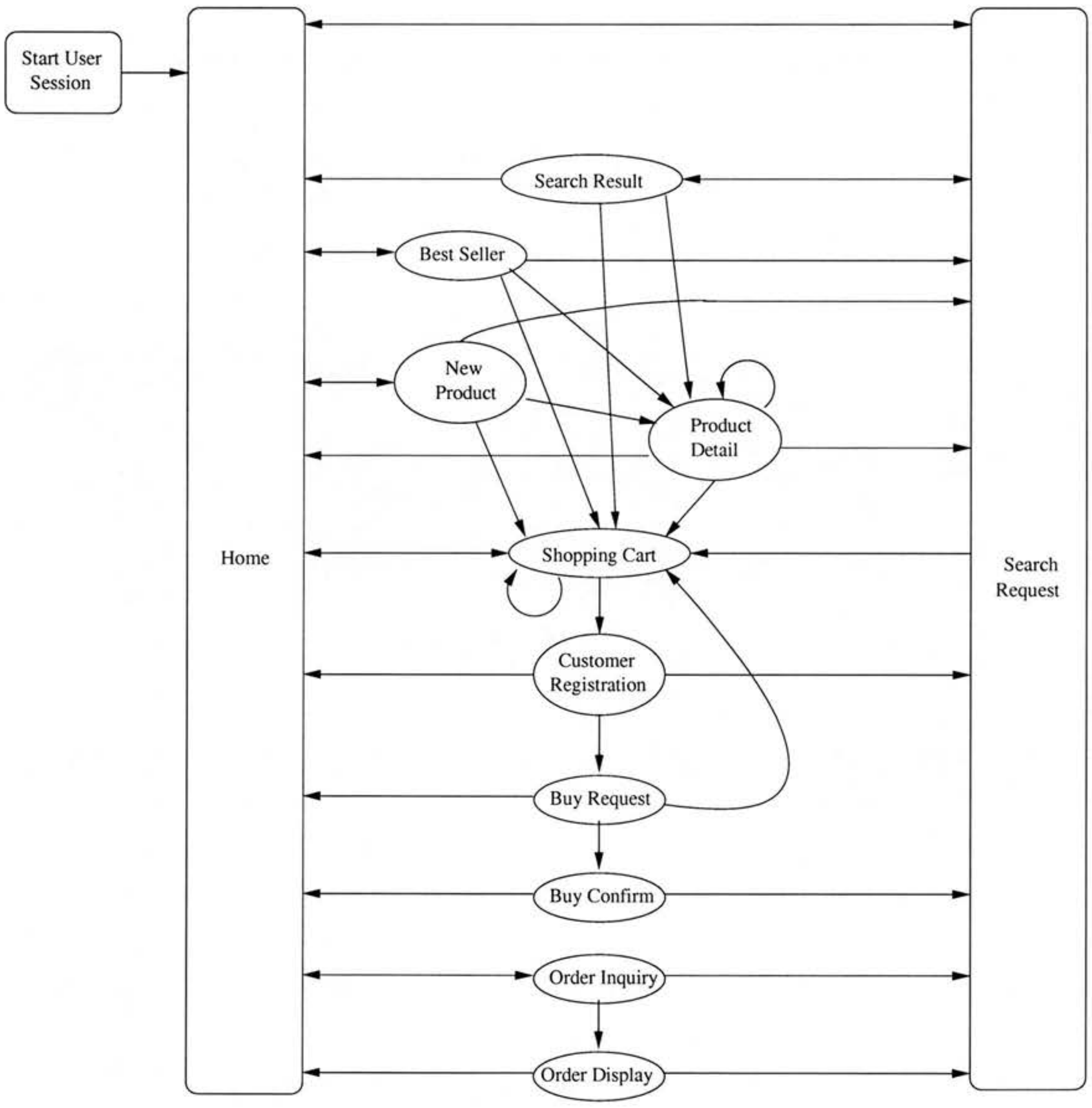


Figure 3.9: TPC-W's Navigational Structure (derived from [3])

- New Products: This interaction returns to the requesting EB a list of items recently published on a specific subject. New Products requires an input parameter *subject* generated randomly by the requesting EB. A database query is executed to search for the 50 most recently published items on that subject. The HTML document is returned contains a list of the items obtained with links to the Product Detail page of each item.

- Search Request: This is a static web interaction which in a real e-commerce site allows the user to enter search criteria in a form. No input parameters are required and no database operations are executed in this interaction.

- Search Result: A list of items that match a search criterion is returned by this interaction. Two input parameters are required, *search_type* and *search_string*, which are generated randomly by the requesting EB. *search_type* determines whether the search is by the book's author, title or subject. The string to be matched in the search is provided by *search_string*. Given the value of the above parameters, a database query is constructed and executed which returns 50 matching items. The HTML code for the page is then generated containing a list of links to the matching items' Product Detail pages.

- Product Detail: This web interaction returns information on a specific item. The item id (*i_id*) is the only input parameter required by this interaction. A database transaction is used to return details of the selected *i_id* including book title, author name, publishing date, subject, cost, availability, image for item, etc. The HTML code for the page is then generated containing the item details.

- Shopping Cart: The cart associated with the shopping session of the user is updated in this interaction. If there is no cart associated with the shopping session, a new cart is created. New items can be added to the cart or existing items can be updated. A list of (*i_id, Qty*) pairs are required as input parameters to Shopping Cart which are pairs of an item to be added to the cart or updated and its quantity. *add_flag* is another input parameter which is required to specify whether the items in the list are to be added or updated in the cart. If *add_flag* has value Y and the item list is not empty, a database transaction is executed to retrieve the details of each item in the pair item/quantity list. The details of each item is stored in the cart together with the item quantity required. However, if *add_flag* has value N and the item list is not empty, the quantities of the items already stored in the cart which match items in the item/quantity list are updated with the new quantities. If the quantity in an input parameter item pair has value 0, the corresponding item is removed from the cart. The HTML document returned contains the updated cart.

- Customer Registration: This interaction returns a web page which is used to register a new user or to authenticate a known customer. There are no input parameters required by this interaction. The web page returned contains a form similar to one that would be filled in by a new user giving their personal details. If the EB requesting the interaction represents a known customer, only a user name and a password would be required.

- Buy Request: This interaction registers a new user or confirms the details of a known customer. One of the input parameters required by Buy Request is *returning_flag*, distinguishing whether the EB making the request represents a *Customer* or a *New User*. If a *Customer*, two additional input parameters are required *uname* and *passwd* which are generated randomly by the requesting EB. A database query is used to check that the *Customer*'s user name and password are correct. However if the requesting EB represents a *New User*, input parameters are required representing the personal details of the user. The values of such parameters are generated randomly by the requesting EB according to the TPC-W specification. A database transaction is used to add a new customer record in the database *Customer* table (see Section 3.8.1). A user name and password are generated randomly by the SUT. The Cart associated with the shopping session is updated with the *New Customer*'s personal details. The HTML code for the returned page is then generated containing the customer details including billing address, the items the customer added to the Cart and details of the payment required. For a *New User*, the newly generated user name and password are also provided. In this page, the user will be asked to confirm that all the details are correct, to provide a shipping address and to give credit card details for payment.

- Buy Confirm: In this interaction, an order is constructed from the content of the Cart associated with a registered user shopping session. The input parameters required include credit card details and the customer shipping address which are generated randomly by the requesting EB. The customer personal details, the items stored in the Cart associated with the customer shopping session, the shipping address and the credit card details are used to construct a database transaction which creates an order record in the Order table (see Section 3.8.1). For each item in the Cart, a record is added to the Order_line table (see Section 3.8.1). The order details are then used to build the HTML document returned.

- Order Inquiry: The page returned by this interaction contains a form which allows a returning customer to enter a user name and a password for authentication before viewing the customer's last order. No input parameters are required by this interaction.

- Order Display: This interaction returns a web page which displays information on the status of the last order placed by the customer represented by the requesting EB. Two input parameters are required, *uname* and *passwd* which are provided by the requesting EB. A database query is used to authenticate the user by finding a match for the *uname* and *passwd* in ECBench database's customer table (see Section 3.8.1). If a match is found, another database transaction is executed to return the details of the last order placed by the customer. The order details are then added to the HTML document returned.

3.7.2 Controlling the Web Server

The configuration of the Web Server is done through the Web Server Control sub-component shown in Figure 3.8. Web Server Control acts as an interface to ECBench's Control Unit (see Section 3.5.5) through which instructions are passed to the Web Server.
The Sampling Clock subcomponent of Web Server Control is responsible for controlling the logging of measurements given the performance metrics, measurement interval structure and the sampling rate sent by the Control Unit.

3.8 Database Server

The server side of an e-commerce on-line store typically has a database server containing a database populated with data such as the store catalogue, customer details and information on the orders placed. The database server interacts with the site's web server receiving queries and updates, and sending back data.

ECBench emulates the database server component through its Database Server illustrated in Figure 3.10. It consists of a DataBase Management System (DBMS) which contains a populated database and a Database Server Control component. The latter is responsible for controlling the server and populating the database.

The DBMS and the design of the database is described in Section 3.8.1. This is followed in Section 3.8.2 by a description of database control and population.

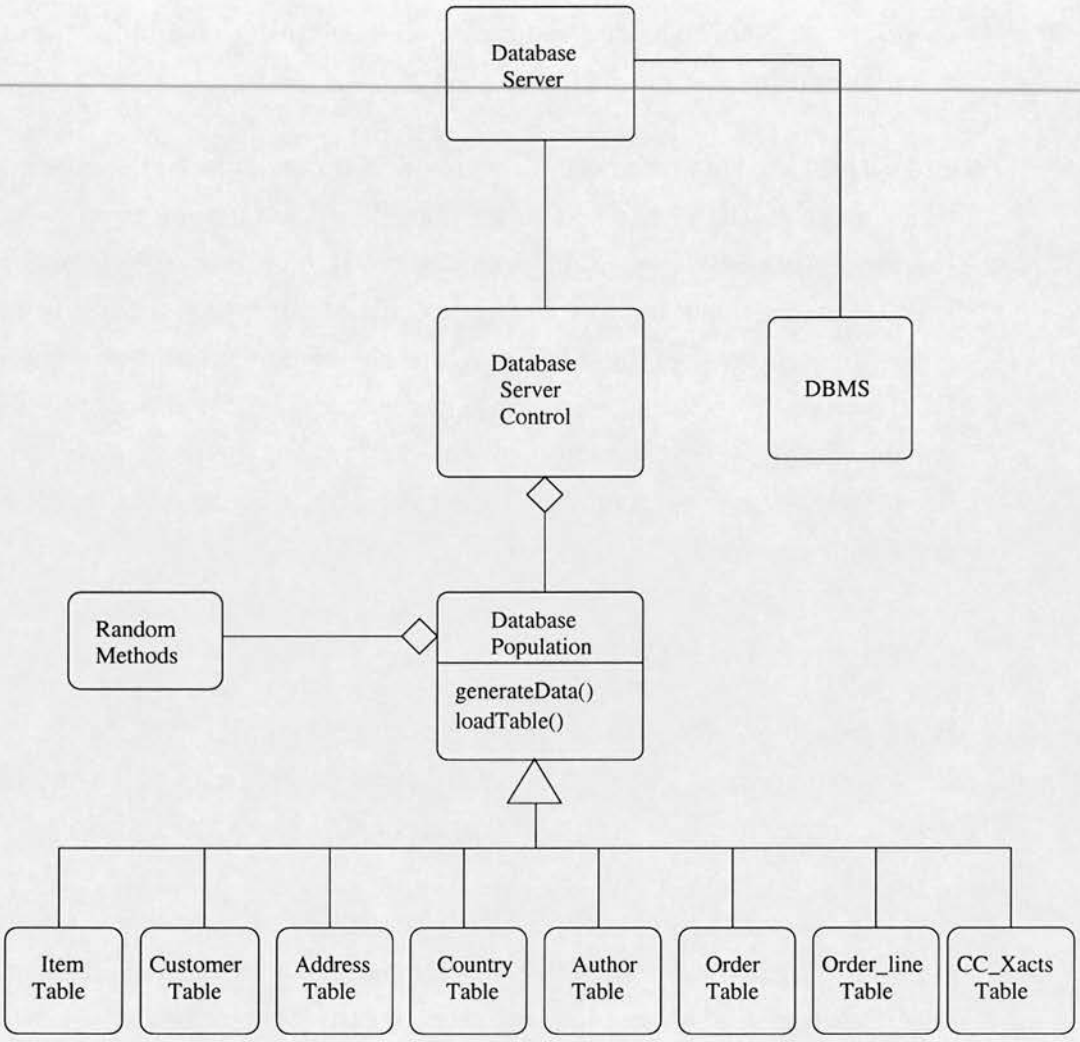


Figure 3.10: ECBench's Database Server

3.8.1 DBMS

The DBMS component of the Database Server hosts the bookstore database which follows the schema specified in the TPC-W specification. Figure 3.11 illustrates the entities composing the database and the relationship between them. The database consists of eight tables, summarised in Table 3.2, *Item, Author, Customer, Orders, Order_line, Address, Country* and *CC_Xacts*. A description of the fields of each table is provided in Appendix A.

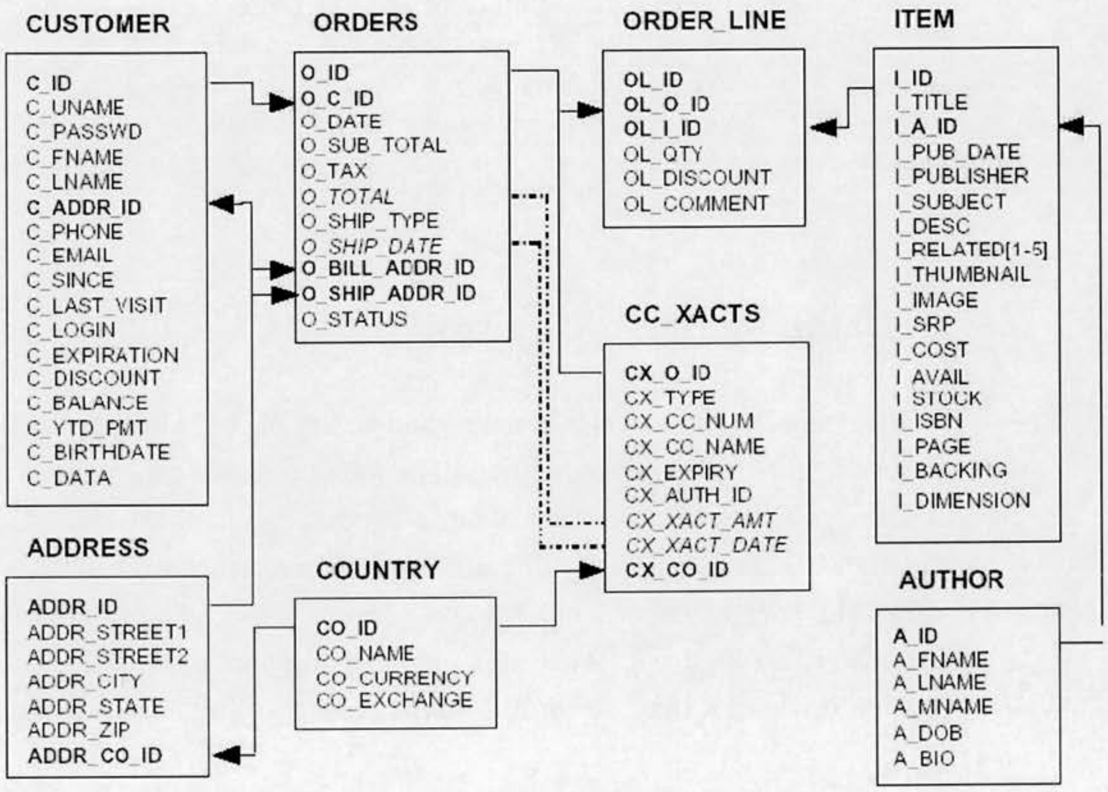


Figure 3.11: Database Entity-Relationship Diagram, from the TPC-W Specification [3]

Table	*Description*
Item	Maintains information on the books offered for sale by the store
Author	Stores details of book authors
Customer	Records registered customers information
Orders	Contains general information on customer orders
Order_line	Includes a break down of the items contained in each order
Address	Stores customers' billing and shipping addresses
Country	Contains different countries details
CC_Xacts	Records customers' credit card details

Table 3.2: Bookstore Database Tables Summary

Table	*Size*
Item	Scale Factor
Author	0.25 * Number of rows in *Item*
Customer	2880 * Workload size
Orders	0.9 * Number of rows in *Customer*
Order_line	3 * Number of rows in *Orders*
Address	2 * Number of rows in *Customer*
Country	92 rows
CC_Xacts	1 * Number of rows in *Orders*

Table 3.3: Bookstore Database Table sizes

3.8.2 Database Server Control

The control of the Database Server is the responsibility of the Database Server Control component illustrated in Figure 3.10. The latter receives control instructions from ECBench's Control Unit as described in Section 3.5.6. Such instructions include the information required to populate the database which are: the size of the workload of the current experiment run and a *scale_factor*. The Database Population unit shown in Figure 3.10 uses these two parameters to determine the size of each table in the database according to TPC-W specification as summarised in Table 3.3.
The RandomMethods library shown in Figure 3.10 implements random functions specified in TPC-W for producing the values of the fields in the database tables. A summary of these functions is given in Appendix B. Sub-components inheriting from Database Population are responsible of generating the data required to populate individual database tables. They use the RandomMethods library to generate values of each field in the tables they are responsible for. The data generated is stored in files and then loaded into database tables.

3.9 ECBench Implementation Issues

In this section, ECBench implementation issues are presented. Firstly, the decision to implement the tool in the Java language is discussed in Section 3.9.1. The use of Apache as the web server of the tool is then explained in Section 3.9.2. Finally, in Section 3.9.3 the choice of MySQL to manage the bookstore database is considered.

3.9.1 Implementation in Java

ECBench is implemented in Java (SDK Version 1.4.2) [47]. Being a pure object-oriented programming language, Java naturally ensured the realisation of the tool's modular design. The implementation in a portable programming language like Java resulted in a platform-independent tool which can be deployed on machines of different architectures across a network. Java comes with many packages which could be exploited during the development of the tool. Java's `java.net` package, with its URL and socket classes, provided a neat implementation of the interactions between the EBs (see Section 3.6) and the Web Server (see Section 3.7) components of the system. Java's `Swing` and `AWT` packages are used to construct ECBench's experimental setup GUI. `Chart2D` [48], a library written in Java to construct 2D graphs is used in ECBench to present experiment results. The reliable `System.currentTimeMillis()` method, which is part of Java's `java.lang` package is used to get timestamps from various parts of the system needed to produce performance metrics. `System.currentTimeMillis()` is quick with almost no overhead, thus enhancing the realistic nature of ECBench experiments.

3.9.2 Apache Web Server

Apache [49] is used as the web server of ECBench (see Section 3.7). It is a popular high-performance freeware web server. Figure 3.12, is a graph from Netcraft [4] summarising the result of surveying the market share of top web servers. The graph shows how Apache has been gaining popularity since 1995 compared to other web servers. Popularity and cheapness were not the only reasons for choosing Apache as the ECBench's web server. The way Apache is designed is another reason. It is built around an API which allows third-party programmers to add new server functionality. Everything in Apache is implemented as one or several modules, using the same extension API available to third parties.
Apache supports many e-commerce technologies where a module is implemented for each technology making it a good choice to widen the range of technologies that can be benchmarked by ECBench. Extending ECBench to support a new e-commerce technology requires just adding a new module to Apache.

3.9.3 MySQL DBMS

MySQL [50] is used as the DBMS of ECBench. It is a free and popular DBMS and works well with Apache for different e-commerce technologies. It hosts the bookstore database described in Section 3.8.

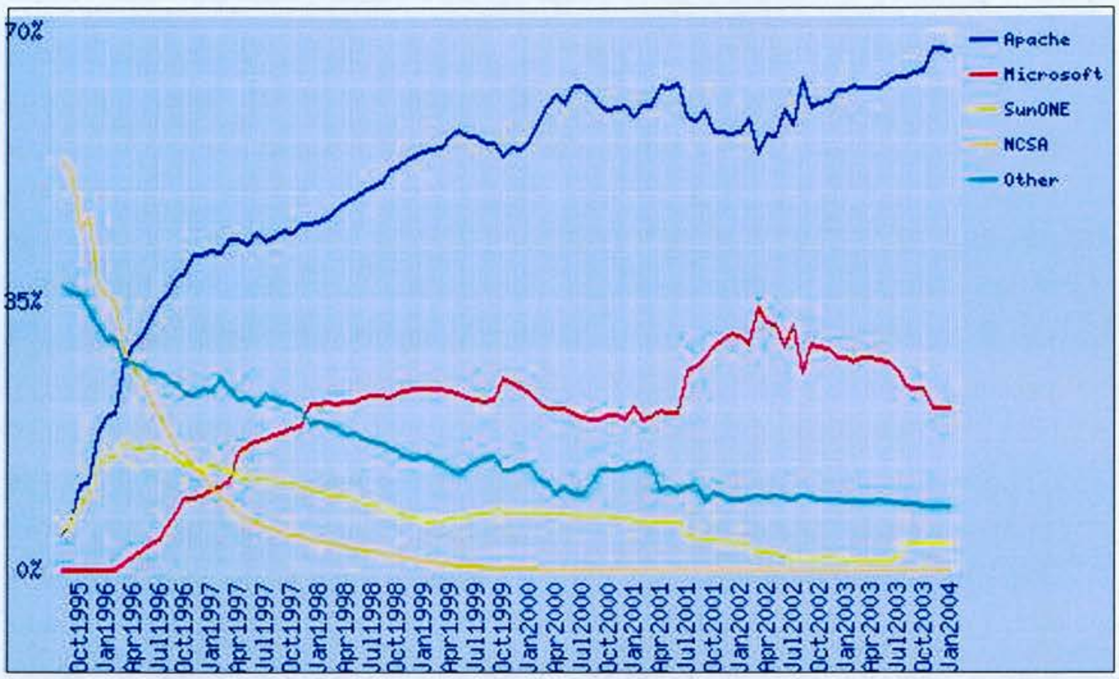


Figure 3.12: The Market Share of Apache, from Netcraft [4]

3.10 Summary

In this chapter, ECBench, a tool to aid the performance benchmarking of Internet Commerce technologies and techniques has been described. The tool is based on the standard TPC-W specification emulating an on-line bookstore. The design of different parts of ECBench were examined including its controlling component, workload, web server and database server. Issues related to the implementation of ECBench were also considered.

In the next chapter, the use of ECBench to benchmark application server programming technologies is discussed. Specifically, PHP and Java Servlets are considered.

Chapter 4

Benchmarking Application Server Technologies

4.1 Introduction

A vital component in an e-commerce system is its application server. As explained in Section 2.4.4, the application server is a software component that provides the business logic of an e-commerce system. It enables the interaction between the web server and the database/transaction server of the system, where data is manipulated and then presented by dynamically generating HTML pages.

Today, many programming environments exist for implementing application servers, allowing the connection between web front-ends and databases and fulfilling the need for dynamic HTML page generation. These include the Common Gateway Interface (CGI) [51], Active Server Pages (ASP) [52], ColdFusion [53], Java Server Pages (JSP) [54], Java Servlets [55] and PHP Hypertext Preprocessor (PHP) [56].

As explained in Chapter 3, ECBench is a general-purpose tool that enables the performance benchmarking of different e-commerce technologies. In this chapter, the use of ECBench to comparatively benchmark PHP and Java Servlets application server technologies is explained. The technologies were incorporated into the tool and experiments were designed to investigate the scalability of the technologies. Experiments were performed and results obtained were analysed using ECBench. This work provides a demonstration of how an e-commerce application developer can use ECBench to comparatively investigate the performance of alternative Internet Commerce technologies. Based on that, the developer can make a decision on the technology that will achieve optimal performance given the system requirements.

The rest of this chapter is structured as follows. In Section 4.2, an overview of PHP is given. This is followed in Section 4.3 by a description of Java Servlets. The benchmarking of PHP and Java Servlets using ECBench is described in Section 4.4. Experiment setup to compare the scalability of PHP and Java Servlets is explained in Section 4.5 and analysis of results obtained is provided in Section 4.6. Finally, in Section 4.7, a chapter summary is given.

4.2 PHP Overview

PHP [56][57] is an open source, server-side, cross-platform, HTML embedded scripting language. It was developed in 1994 by Rasmus Lerdorf to keep track of visitors to his on-line resumé. Since then, it has undergone several changes with two versions released PHP3 and PHP4. PHP originally stood for *Personal Home Pages* but the name was later altered to *PHP Hypertext Preprocessor*. As of June 2004, more than sixteen million domains worldwide use PHP [58].

PHP can be deployed to run as a server-side module in a web server such as Apache. Embedding PHP code into HTML documents enables the generation of HTML code dynamically. PHP borrows many concepts of common languages such as C and Perl including syntax, datatypes and control structures. Many web programming features are provided by PHP including form parameter handling and session tracking techniques. Global arrays are created to hold HTTP request details including form parameter and cookie values. These arrays are accessed in the PHP script to fulfil the request.
PHP supports session tracking (see background Section 2.4.4) in two ways: using cookies or a more complex built-in system. For cookies, PHP allows the user to set different parameters in a cookie including name, value and expiry date. Cookie details are retrieved using the global arrays explained above. PHP also has built-in support for session tracking by providing a unique session ID for every new visitor. The session ID is stored in a cookie called PHPSESSION which is persistent across user requests. Variables can be associated with the session ID and stored in the session's allocated space.

Many relational Database Management Systems (DBMS) are supported by PHP including MySQL, PostgreSQL and Oracle. Databases can be accessed in two ways using PHP. The first is to use a database-specific extension with function names, parameters and error handling specific to the required database. The other way is to access the required database using the PEAR DB library which comes with PHP. PEAR DB is a database-independent library which hides database specifics allowing the same PHP script to access many DBMSs. One disadvantage of the PEAR DB method is that it can be too general, failing to support features provided by a particular database. Another disadvantage of PEAR DB code is that it can be slower than code written in database-specific extensions.

A sample PHP HTML embedded script is shown in Figure 4.1. The script allows a customer to query the details of an item in a bookstore database given the item ID. The example uses many of the concepts described earlier. PHP's built-in session tracking system is used to keep track of the number of visits made by the customer. The $_GET global variable is used to get the customer request's itemID parameter. MySQL's extension functions are used to connect to the bookstore database and query its item and author tables to get the item details. HTML code is then generated which includes the data returned from the database query.

4.3 Java Servlets Overview

Java Servlets [55] are server-side Java code that are used to extend server capabilities. Since servlets are written in the highly portable Java language and follow a standard framework, they provide a means to create sophisticated server extensions in a server and operating system independent way. Servlets are commonly used to support HTTP requests in web applications implementing the application server layer between a web server and a database server and providing dynamic content for user requests. They can also be used in mail and FTP servers. Java Servlets are part of the Java 2 Platform, Enterprise Edition (J2EE) [59] and the Apache web server [49].

Figure 4.2 illustrates the hierarchy of HTTP Java Servlets. They are constructed using classes in javax.servlet and javax.servlet.http Java standard extension packages. All HTTP servlets must inherit Java's HTTPServlet class which is a subclass of GenericServlet. The latter implements the Servlet interface.
Methods to control the life-cycle of a servlet are provided by the Servlet interface. Figure 4.3 illustrates the life-cycle of a HTTP servlet ServletA deployed in a server. When a client makes a request to ServletA, the server checks whether an instance of the servlet exists or not. If no instance is found, the server loads

```
<?php
    // Starting/loading user session to keep track of number of visits
    session_start();
    session_register('visits');
    ++$visits;

?>

<HTML><HEAD><TITLE>
PHP Item Details Page
</TITLE></HEAD>
<BODY>
<H1> Item Details Page </H1>

<?

   // get itemID parameter contained in the request.
   $itemId = $_GET['itemID'];

   // Get a Connection to the bookstore database.
   $DBhost = "localhost";
   $DBuser = "root";
   $DBName = "bookstore";
   mysql_connect($DBhost, $DBuser);
   mysql_select_db($DBName);

   // Execute a sql query getting item information from item and author tables.
   $query = "Select i_title, a_fname, a_lname from item, author
                                    where i_id= $itemId and i_a_id = a_id";
   $result = mysql_query($query);

   // Generate HTML code containing number of visits made by
   // customer and item information.
   echo "<H3>Number of Visits: ", $visits, "</H3>";
   echo "<H3>Item ID: ", $itemId, "</H3>";
   echo "<H3>Book Title: ", mysql_result($result, 0, 'i_title'), "</H3>";
   echo "<H3>Author First Name: ", mysql_result($result, 0, 'a_fname'), "</H3>";
   echo "<H3>Author Last Name: ", mysql_result($result, 0, 'a_lname'), "</H3>";
   mysql_close();
?>

</BODY>
</HTML>
```

Figure 4.1: PHP Sample Code

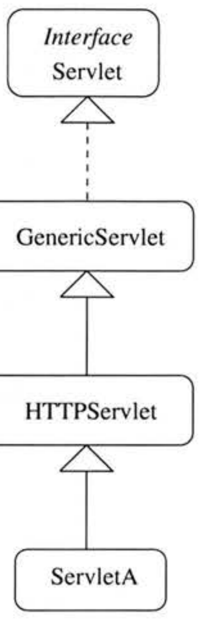


Figure 4.2: HTTP Java Servlet hierarchy

the ServletA class and instantiates it to create an instance of the servlet. The server then initialises ServletA instance by calling the init() method of the Servlet interface. ServletA should override the init() to customise the initialisation process. The service offered by a HTTP servlet should be implemented by overriding the appropriate doMethod (e.g. doGet(), doPost()) of the GenericServlet class. To fulfil the client request, the appropriate service method is called on ServletA. A response is then constructed and returned to the client. Subsequent requests to ServletA do not require its class to be loaded or the init() method to be called. The service method of the servlet is called directly. If the servlet needs to be removed, e.g. on server shutdown, the destroy() method of the Servlet interface is called. ServletA should override this method if operations such as the release of system resources are required on termination of the servlet.

Java HTTP servlets enable session tracking (see background Section 2.4.4) in two ways, using cookies or a built-in session tracking system. Support for cookies is provided in the Cookie class which is part of the javax.servlet.http package. It contains methods to set and retrieve various cookie parameters. Java Servlets' built-in session tracking mechanism is provided by the HttpSession interface of javax.servlet.http. HttpSession is used by the server containing a HTTP servlet to create a session and associate it with a particular user. Object-valued attributes can be associated with a session by name and can also be retrieved.

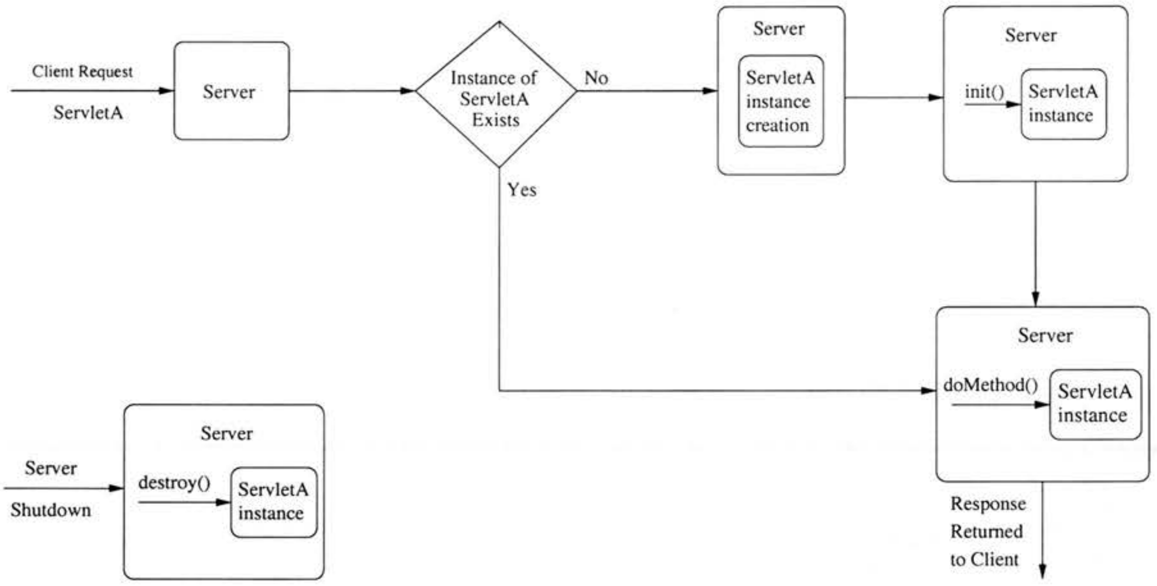


Figure 4.3: HTTP Java Servlet Life-cycle

Java Servlets offer excellent connectivity with many DBMSs. This is ensured by using Java Database Connectivity (JDBC) which enables the creation of database-independent Java applications. Using JDBC, Java Servlets can communicate with any DBMS as long as there is a JDBC driver specific to that DBMS.

The code for a sample Java Servlet ServletExample, which uses many of the features described in this Section, is listed in Figure 4.4. It implements the same item details service as the PHP script described in Section 4.2 and listed in Figure 4.1. ServletExample is a HTTP servlet inheriting the HttpServlet class. It supports HTTP GET requests overriding GenericServlet's doGet() service method. Java Servlets' built-in session tracking mechanism is used to keep track of the number of times the customer requested the servlet. The HTTPServletRequest object parameter of doGet() is used to retrieve the itemID parameter. The JDBC driver for the MySQL DBMS is loaded to connect to the bookstore database and get the item details. All HTML code generated is added to the output stream of doGet()'s HttpServletResponse object.

4.4 PHP and Java Servlets Benchmarking

To comparatively benchmark PHP and Java Servlets, both technologies were used to implement ECBench's application server (see Section 3.7.1). Apache version

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletExample extends HttpServlet{

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException{
        int visits = 1;
        PrintWriter out;

        // Starting/loading user session to keep track of number of visits
        HttpSession session = request.getSession(true);
        if(session.isNew()){
            session.putValue("visits", String.valueOf(visits));}
        else{
            visits = Integer.parseInt( (String) session.getValue("visits"));
            session.putValue("visits", String.valueOf(++visits));
        }

        response.setContentType("text/html");
        out=response.getWriter();
        out.println("<HTML><HEAD><TITLE>");
        out.println("Servlet Item Details Page");
        out.println("</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1> Item Details Page </H1>");

        try{
            // get itemID parameter contained in the request.
            String itemId = request.getParameter("itemID");

            // Get a Connection to the bookstore database.
            Class.forName("org.gjt.mm.mysql.Driver").newInstance();
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/bookstore?user=root");

            // Execute a sql query getting item information from item and author table.
            PreparedStatement query =  conn.prepareStatement("
                    Select i_title, a_fname, a_lname from item, author where i_id=? and i_a_id = a_id");
            query.setInt(1,Integer.parseInt(itemId));
            ResultSet rs = query.executeQuery();

            // Generate HTML code containing number of visits made by customer and item information.
            out.println("<H3>Number of Visits: " + visits + "</H3>");
            out.println("<H3>Item ID: " + itemId + "</H3>");
            out.println("<H3>Book Title: " + rs.getString("i_title") + "</H3>");
            out.println("<H3>Author First Name: " + rs.getString("a_fname") + "</H3>");
            out.println("<H3>Author Last Name: " + rs.getString("a_lname") + "</H3>");
            conn.close();

            out.println("</BODY>");
            out.println("</HTML>");
            out.close();
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Figure 4.4: Java Servlet Sample Code

1.3.31 [49] was used as ECBench's web server (see Section 3.9.2). MySQL version 3.23.45 [50] was used as ECBench's DBMS (see Section 3.9.3). Modules were added to the Apache server to provide support for the PHP and Java Servlets implementations of the application server.

In Section 4.4.1, the PHP application server is described. This is followed in Section 4.4.2 by a description of the Java Servlets application server.

4.4.1 Benchmarking PHP

PHP 4.1.0 [60] was used in the implementation of the PHP application server. It was configured to run as a module for ECBench's Apache web server and to communicate with ECBench's MySQL DBMS. A PHP embedded HTML script was constructed for each of the web interactions supported by ECBench and described in Section 3.7.1. The scripts follow the sample code listed in Figure 4.1 and described in Section 4.2.

4.4.2 Benchmarking Java Servlets

Apache's JServ [61] (version 1.1.1) was used to incorporate the Java Servlets application server implementation into ECBench's web server. Apache JServ is designed using a three-tier model: the Apache web server, the mod_jserv Apache module and Apache JServ server. When the Apache web server receives a HTTP servlet request, it handles it through its mod_jserv module. mod_jserv translates the request into Apache JServ Protocol (AJP) format and passes it on to the Apache JServ servlet engine. Apache JServ is a standalone server implemented in Java. It translates the AJP request into a HTTPServletRequest object, processes it creating a HTTPServletResponse which it passes back in AJP format to mod_jserv. The AJP response is then translated by mod_jserv into a HTTP response and passed back to the client. Separating the web server from the servlet engine avoids the scalability problem that could arise from running the Java Virtual Machine (JVM), needed to handle Java Servlets, in each Apache process.
A servlet was constructed for each of the web interactions supported by ECBench and described in Section 3.7.1. All of the servlets implemented follow the ServletExample code listed in Figure 4.4 and described in Section 4.3.

4.5 Experiment Setup

An experiment was designed and performed using ECBench to investigate the scalability of PHP and Java Servlets application server implementations. In Section 4.5.1, the design of the experiment using ECBench's design features (see Section 3.5.1) is discussed. This is followed in Section 4.5.2 by a description of how ECBench was deployed during the experiment. Finally, in Section 4.5.3, the way measurements were taken during the experiment is described.

4.5.1 Experiment Design

The aim of the experiment was to analyse and compare the scalability of PHP and Java Servlets technologies. The application server implementations in both technologies were incorporated into ECBench as explained in Section 4.4. ECBench experiment design features (described in Section 3.5.1) were used to setup the experiment's parameters. The independent parameter was the size of ECBench workload which was varied by doubling the the number of EBs at each stage with a minimum workload size of 1 EB and a maximum size of 1024. Thus, 11 workload size levels were examined: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 and 1024. The scalability of PHP and Java Servlets was analysed by examining the effect of workload size levels on different performance metrics including throughput, number of failed requests, ECBench server processor utilisation, average response time and individual web interactions average response time. A scale factor value of 1000 was used to determine the size of ECBench's database as described in Section 3.8.2. The total time duration of each experiment run was 400 seconds. An initial *Ramp up* period of 200 seconds was used to allow for the workload to reach a steady state. Initial experiments showed that 200 seconds is enough to allow for all EBs composing the workload to become fully functional (for all workload sizes ranging between 1 and 1024). The remaining 200 seconds was the duration of the measurement interval. The type of measurement interval was specified as *Shopping* reflecting a shopping scenario (described in Section 3.2.4). A sampling rate of one minute was used to record system measurements.

Two sets of experiment runs were performed, one using the PHP implementation of ECBench's application server and the other using the Java Servlets implementation. For each workload level, three experiment runs were replicated averaging the results obtained. Thus, a total of 66 experiment runs were performed in this experiment.

4.5.2 ECBench Deployment

Initial experiments showed that having all EBs at high workload size on one machine caused the results obtained to be dominated by the delay occurring from the client preparing and making requests and receiving returned HTML documents from the server. Having a large number of EBs on one machine also contradicts a real system scenario where only one request is usually made at a time per client machine. To keep the client processing delays to a minimum and to be as realistic as possible, in high workload size runs the EBs were distributed on multiple machines as summarised in Table 4.1. Each of the machines was running Linux and had an Intel Pentium 4, 1.80GHz processor.
The Web Server component of ECBench including the PHP and Java Servlets application servers were deployed on an Intel Pentium III 1GHz Linux machine. The Database Server component of ECBench was deployed on the same machine.

Workload Size	*1*	*2*	*4*	*8*	*16*	*32*	*64*	*128*	*256*	*512*	*1024*
Number of Machines	1	1	1	1	1	1	2	4	8	16	32

Table 4.1: Workload Deployment

4.5.3 System Measurements

As described in Section 3.9.1, ECBench uses the `System.currentTimeMillis()` method, which is part of Java's `java.lang` to get timestamp measurements. This feature was used in the experiment to get measurements for response time metrics.

As ECBench was deployed on Linux machines during the experiment, the */proc/stat* file in Linux's *proc* filesystem [62] was used to get measurements needed to calculate processor utilisation on the ECBench server. The */proc/stat* file contains statistics on the number of jiffies (1/100 second) that the machine's cpu was in user mode, user mode with low priority (nice) and system mode, since the machine was booted. Using a sampling rate of one minute, the */proc/stat* file was read and the cpu jiffies were added. Subsequent measurements were subtracted and the result was divided by 6000 to get the processor utilisation per minute.

4.6 Results Analysis

ECBench features, discussed in Section 3.5.2, were used to gather and analyse results obtained from performing the experiment described in Section 4.5. As explained, the aim of the experiment was to analyse the scalability of PHP and Java Servlets application server implementations. On the completion of each experiment run, data measurements were gathered from different parts of ECBench. At the end of the experiment, run results were averaged and analysed for each of the experiment's metrics. Graphs were produced, visually combining PHP and Java Servlets results.

The results with respect to different performance metrics are discussed in the following subsections.

4.6.1 Effect of Varying Workload Size on Throughput

The throughput metric is considered as the total number of web interactions requested and completed successfully during the measurement interval. The reader should note that throughput is not considered as a rate in this context. Table 4.2 and Figure 4.5 summarise the results obtained from examining the throughput of PHP and Java Servlets when the size of the workload is varied between 1 and 1024. It can be seen that for both PHP and Java Servlets throughputs start

Workload Size	*PHP*	*Java Servlets*
1	901	1053
2	528	599
4	366	420
8	386	379
16	389	388
32	402	446
64	844	854
128	1159	1676
256	3045	3140
512	5934	3305
1024	11493	3209

Table 4.2: Throughput Metric (Total Number of Requests Completed Successfully)

slightly high at workload size 1. This is due to the lack of contention as the servers are handling one request at a time generated by the one EB making the workload. The throughput starts to drop as the workload size increases due to the contention delay. At workload size 16, the throughput starts to increases slightly as the rate of requests handled by the servers becomes more dominant than the contention effect. At workload size 64, a greater increase rate is noticed for PHP and Java Servlets throughputs. This is due to the distribution of the

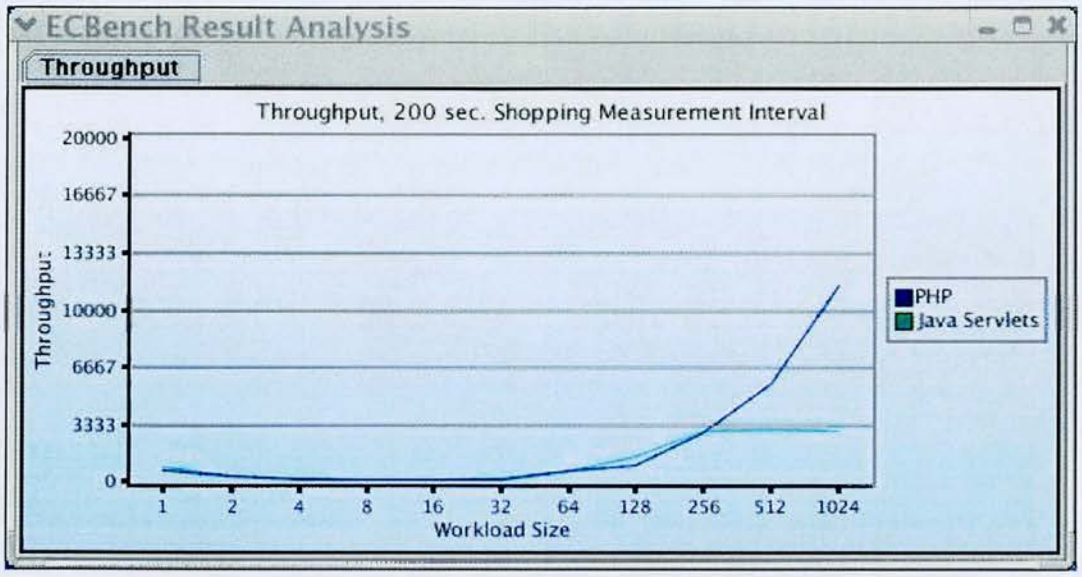


Figure 4.5: Workload Size vs. System Throughput

workload across more than one client machine as described in Section 4.5.2. At this stage, the client overhead caused by the generation of requests and handling of returned web pages becomes constant and the effect of increasing rate of request on achieving more throughput becomes clearer. The throughput for PHP continues to increase but drops for Java Servlets due to the reason described in Section 4.6.6.

4.6.2 Effect of Varying Workload Size on Request Failure

An EB web interaction request is considered as a failed request if the EB receives an error from ECBench's web server in response to that request. The request failure metric represents the total number of failed requests in the measurement interval. The request failure results at different levels of workload size are shown in Table 4.3 and Figure 4.6. At low workload sizes, both PHP and Java Servlets implementations provide failure-free support for web interaction requests. At high load, the Java Servlet implementation suffers severely, especially at the 1024 workload size. The PHP implementation starts returning a few failed requests at the 1024 workload size, but still significantly less than the Java Servlets implementation at the 512 workload size.

Workload Size	*PHP*	*Java Servlets*
1	0	0
2	0	0
4	0	0
8	0	0
16	0	0
32	0	0
64	0	0
128	0	0
256	0	0
512	0	45
1024	4	454

Table 4.3: Request Failure Metric Results

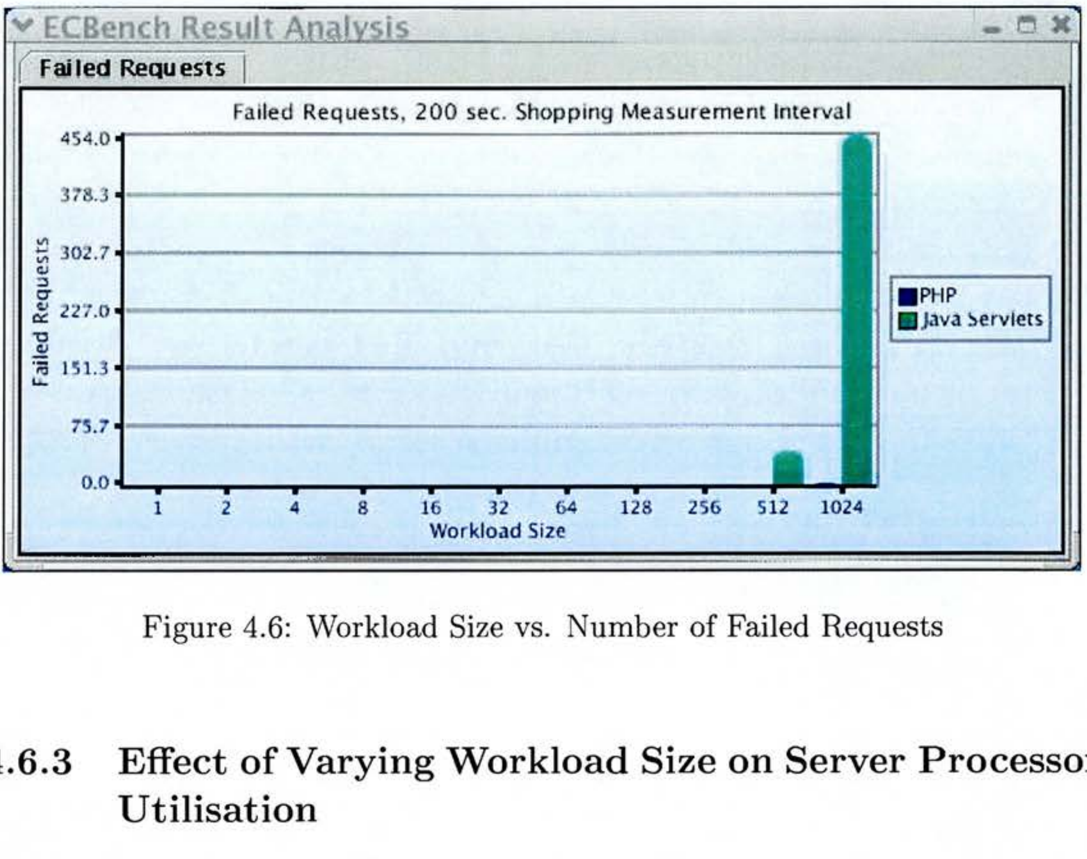


Figure 4.6: Workload Size vs. Number of Failed Requests

4.6.3 Effect of Varying Workload Size on Server Processor Utilisation

The server processor utilisation metric is considered as the average processor utilisation during the measurement interval. As explained in Section 4.5.3, samples of ECBench's server processor utilisation are taken every minute during the measurement interval. The samples (3 samples in this experiment) are averaged to get a value for the metric over each experiment run. Table 4.4 and Figure 4.7 summarise the results obtained from examining the processor utilisation of the ECBench server when the size of the workload is varied. The processor utilisation

Workload Size	*PHP*	*Java Servlets*
1	0.27	0.177
2	0.1	0.12
4	0.075	0.11
8	0.077	0.084
16	0.072	0.098
32	0.075	0.102
64	0.13	0.22
128	0.23	0.36
256	0.48	0.71
512	0.897	0.983
1024	0.997	0.999

Table 4.4: Processor Utilisation Metric Results

makes a slightly high start at workload size 1 for both PHP and Java Servlets utilisation, due to the lack of database contention with one EB. This result is consistent with the higher throughput readings at workload size 1 explained in Section 4.6.1. At workload size 2, processor utilisation decreases for both PHP and Java Servlets reflecting the effect of contention. The processor utilisation then stays at almost a constant level until workload size 32. As the workload size increases more and the EBs are distributed across more than one client machine, the effect of increasing the request rate causes the utilisation to increase steadily for both PHP and Java Servlets with the latter causing a higher utilisation. At

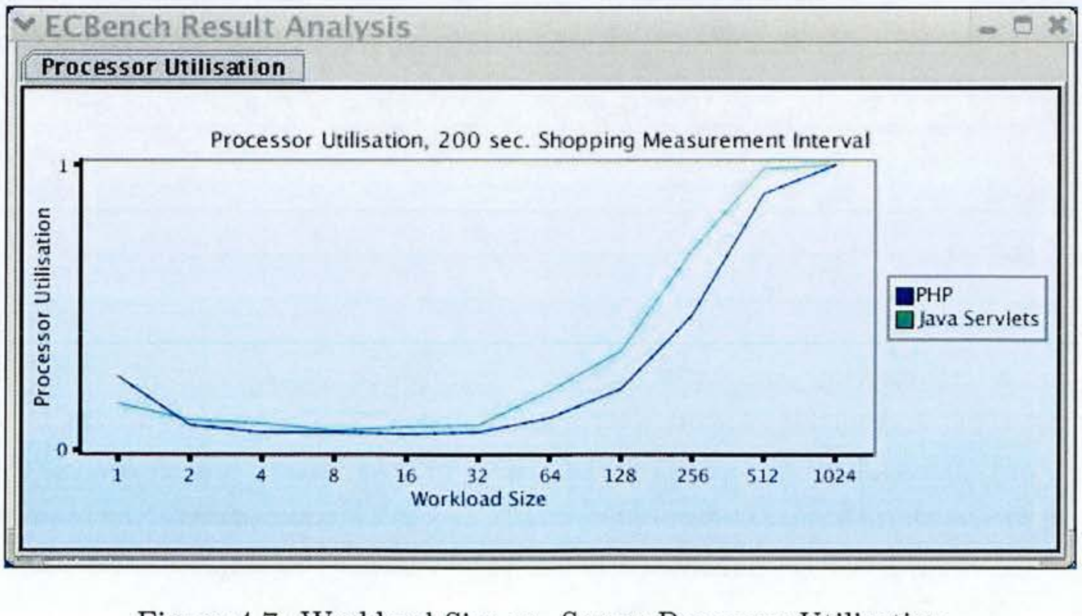


Figure 4.7: Workload Size vs. Server Processor Utilisation

the 1024 workload level, both PHP and Java Servlets utilisations are approaching 1 with Java Servlets again slightly higher.

4.6.4 Effect of Varying Workload on Average Response Time

The response time of a web interaction is considered as the time elapsed from the first byte sent by an EB to make a request until the last byte of the HTML document received by the EB in response to the request. Thus, response time is affected by:

- The overhead caused by contention on the client due to request generation, sending and handling of web pages returned,
- and server delays in handling requests.

The average response time metric is considered as the total response time of all web interactions completed successfully in the measurement interval divided by the number of these interactions (throughput). Table 4.5 and Figure 4.8 summarise the results obtained from examining the average response time when varying the workload size. At low loads, response time increases steadily and almost following the same line for both PHP and Java Servlets. This steady increase is primarily caused by client overhead due to contention. At workload size 64, response time flatten off due to constant client overhead as EBs were distributed equally across machines (as described in Section 4.5.2). At workload size 256, the response time starts to increase sharply for Java Servlets and at a slower rate for PHP caused by the delays on the server (see Section 4.6.6 for an explanation on the sharp increase of Java Servlets' response time).

4.6.5 Effect of Varying Workload Size on Individual Interactions Average Response Time

The average response time metric examined in Section 4.6.4 considers overall response time for all web interactions without distinguishing between the different types. However, examining web interactions response time on an individual basis could be useful, for example to find the web interaction with the highest response time.

The average response time for an individual interaction is computed by totalling the response time of all occurrences of that interaction and dividing that by the number of times the interaction was requested and completed successfully. Graphs

Workload Size	*PHP*	*Java Servlets*
1	79.1	56.8
2	255.2	241.4
4	775.8	678.0
8	2194.8	2279.9
16	5441.8	5575.7
32	11629.8	11177.0
64	12104.8	12329.4
128	11841.5	11046.1
256	11487.6	11939.5
512	12608.5	23469.5
1024	13643.3	31153.4

Table 4.5: Overall Average Response Time Metric Results

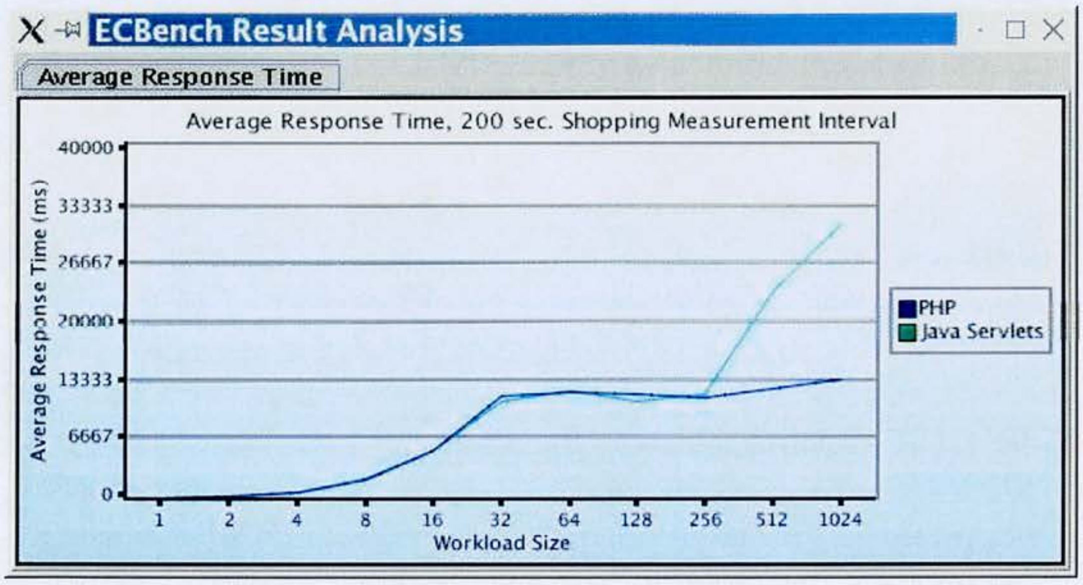


Figure 4.8: Workload Size vs. Overall Average Response Time

of the average response times for each web interaction supported by ECBench are shown in Figure 4.9. Tables summarising the data obtained are included in Appendix D. It can be seen that the overall average response time analysis provided in Section 4.6.4 reflects, to a great extent, the individual web interaction response times. Response time increases gradually, for both PHP and Java Servlets, until the workload size of 32. Then it stays almost constant until workload size of 256. At higher workload sizes, Java Servlets response time for all interaction starts to increase sharply with a much lower increased rate for PHP.

4.6.6 Discussion

It can be seen from the analysis of different metrics that Java Servlets have a scalability problem at high loads. At workload sizes higher than 256, Java Servlets throughput, failed requests and response time start to degrade sharply compared to the much slower degradation in PHP performance. Investigating Apache server logs to find out the cause of the increase in the number of failed requests at high load, I found errors of the form

```
[error] [client 129.215.18.34]
    Premature end of script headers: /servlets/ShoppingCartServlet
```

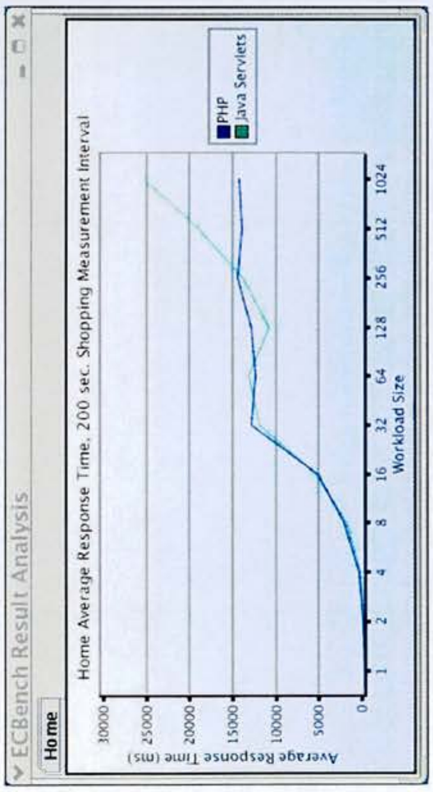
EBs receiving failed responses were getting an *Internal Server Error* page. Investigating the matter further, I found in the java-apache-users mailing list [63] posts of two Java Servlets developers who encountered the same error at high workload size. The JVM of the Apache JServ server (described in Section 4.4.2) hosting the Java Servlets implementation can handle a limited number of objects. High workload size translates into a large number of requests, thus an increase in the number of database objects in the JVM since one object is created per dynamic web interaction request. When the number of database objects exceeds the JVM limit, garbage collection is called and the servlets are re-initialised. At this stage the server can not fulfil requests causing errors to be logged and failed responses to be sent back to clients. This causes the throughput of Java Servlets to drop sharply. The re-initialisation of servlets and the logging of errors consume more CPU cycles which slow the server down and cause the increase in web interaction response times.

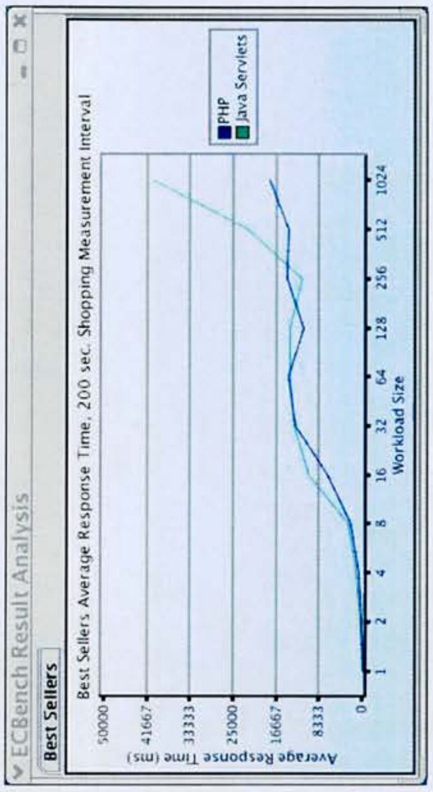
This scalability problem of Java Servlets can be overcome by re-using database objects. This can be achieved by creating a database connection pool where connection objects are created at servlet initialisation and re-used across requests rather than creating a connection per request. Sample code for a servlet that uses connection pooling can be found in [55]. Evaluation of this solution was considered to be beyond the scope of this thesis.

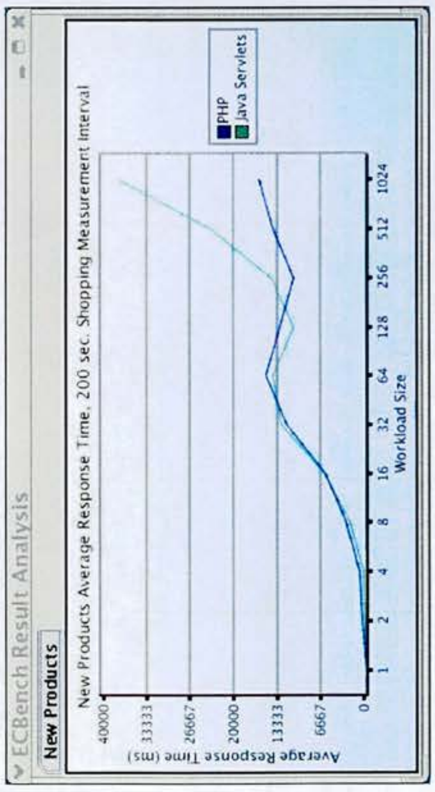
The use of ECBench validates the results described in this chapter. This is because ECBench provides an emulation of a real Internet Commerce system. It is based on the TPC-W specification regarded as the Internet Commerce system standard. Verification was achieved by extensive debugging of ECBench ensuring that the functionality of different parts of the system is correct.

4.7 Summary

The work described in this chapter provides a demonstration of the effective use of ECBench to investigate the performance of different e-commerce technologies. This aids the process of selecting the technology that optimally meets system QoS requirements at early stages of system design rather than facing performance problems when the system is fully implemented and online.
In the next chapter, CBAC is introduced, a novel cost-based admission control policy to control QoS in e-commerce systems.







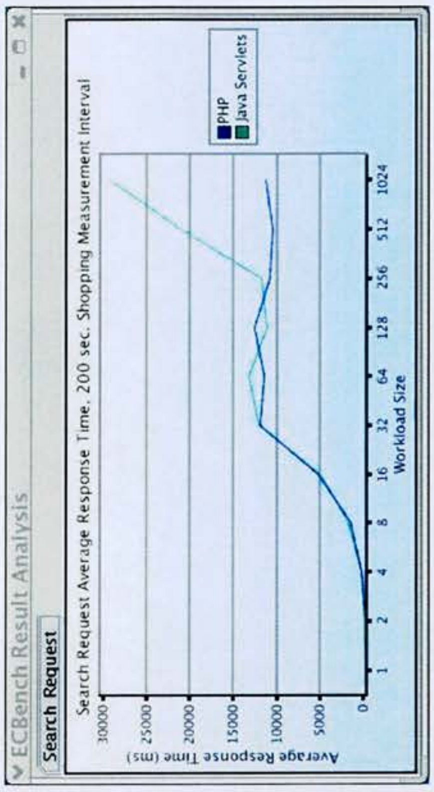
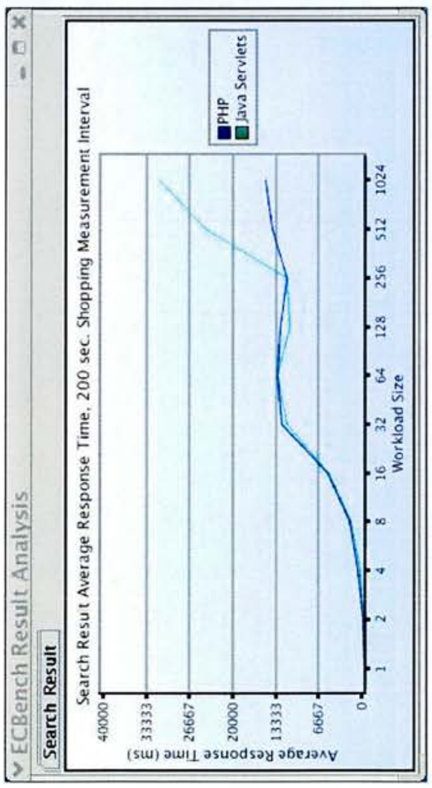
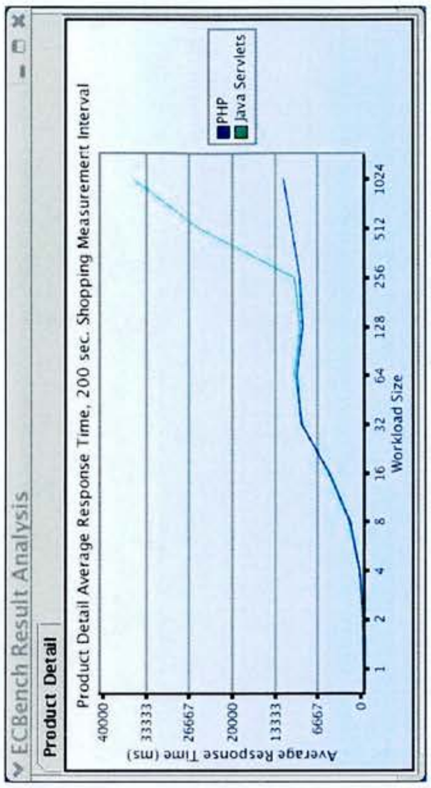
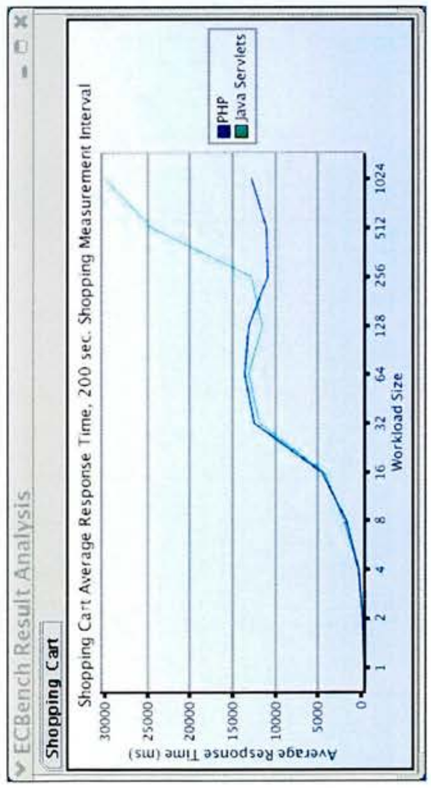


Figure 4.9: Workload Size vs. Individual Interactions Average Response Time (Cont. next page)







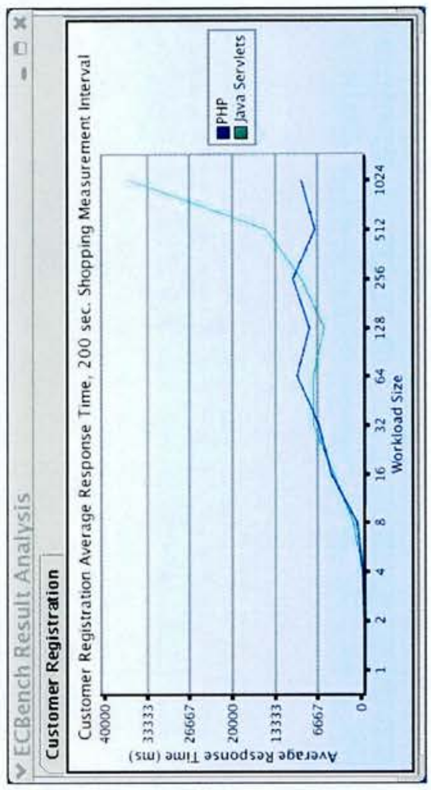
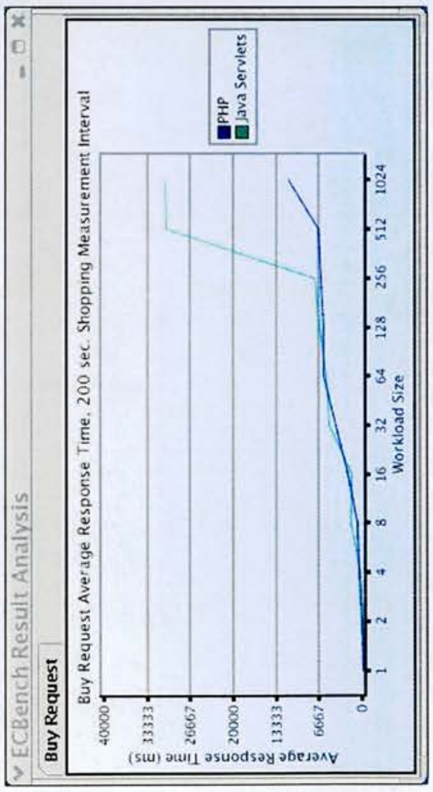
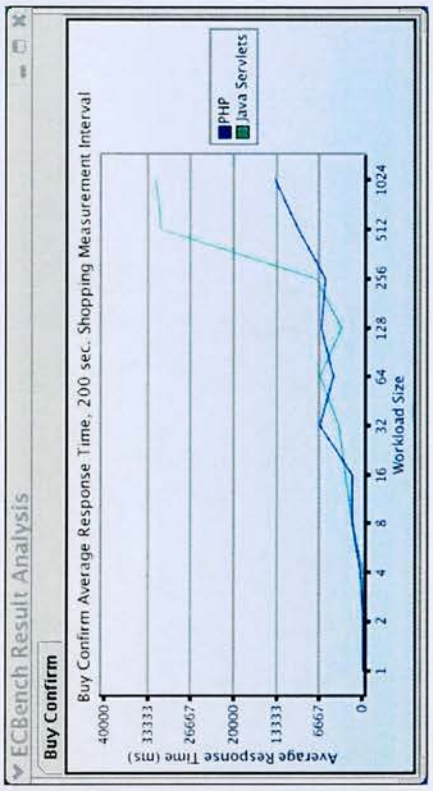
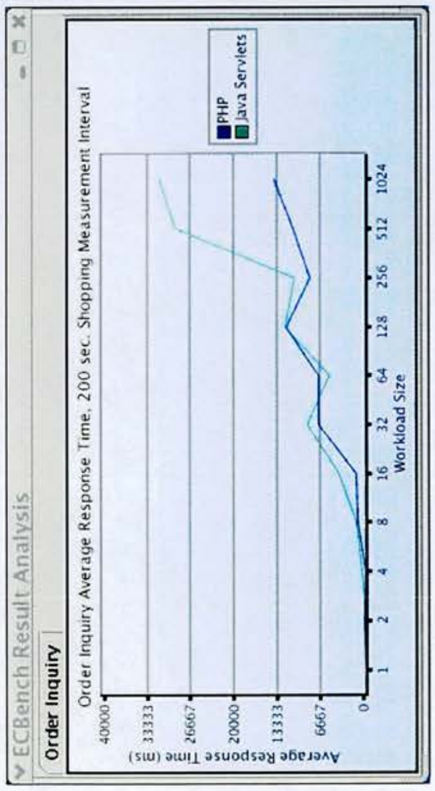


Figure 4.9: (Cont.) Workload Size vs. Individual Interactions Average Response Time (Cont. next page)







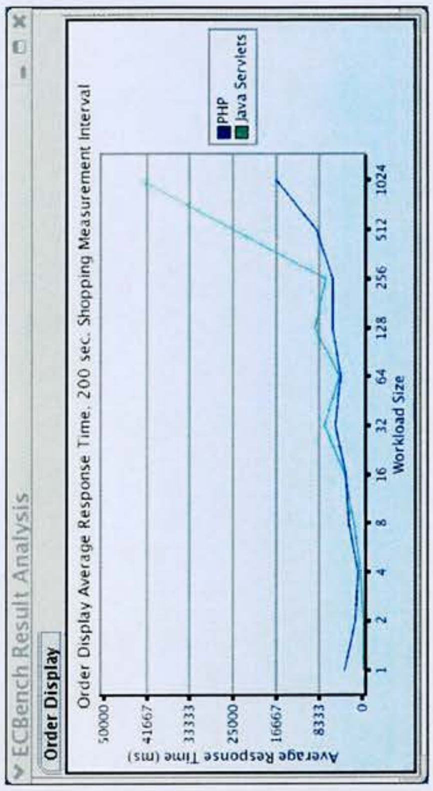


Figure 4.9: (Cont.) Workload Size vs. Individual Interactions Average Response Time

Chapter 5

Cost-Based Admission Control for Internet Commerce QoS Enhancement

5.1 Introduction

QoS enhancement methods can be applied to Internet Commerce systems which are online and fully operating. This dynamic enhancement of QoS is required to complement QoS-oriented decisions made during the system's design stage, given the highly unpredictable nature of Internet performance and the changing characteristics of Internet Commerce workload. Different solutions have been used to allow for this dynamic enhancement of QoS to provide system performance guarantees and differentiated service. Hardware solutions are the most widely used including server replication and caching. Such approaches can be expensive financially and introduce complex technical problems including load balancing and consistency. Software techniques such as web content adaptation and admission control have also been considered. Three disadvantages can be identified in such techniques. Firstly, such techniques usually rely on user request rejection to avoid a system overload state. This undermines customer satisfaction and thus the probability of return. Secondly, they usually consider only one aspect of QoS, either to improve response time, maximise profit or enhance system resource utilisation. A comprehensive framework for QoS enhancement is not considered. Finally, very few techniques have been designed to specifically target Internet Commerce systems, rather they focus on web QoS in general.

In this chapter, a Cost-Based Admission Control approach (CBAC) for Internet Commerce systems is described. It is a software technique to dynamically enhance QoS in an Internet Commerce system while the system is online. Without the need to reject customer requests in a high-load situation a discount-charge model, which is sensitive to system current load and navigational structure, is used to encourage customers to postpone their requests. A scheduling mechanism with load forecasting is used to schedule user requests in more lightly loaded time periods. At high load, the aim of CBAC is to ensure that all main Internet Commerce QoS metrics are considered. It provides service times competitive with low load situations by ensuring that system load is distributed to less loaded future time intervals. The charging element of CBAC and abandoning request rejection, to increase customer satisfaction, is designed to maximise profit. Scheduling customer requests in lightly loaded time periods increases system utilisation, sustains throughput and reduces request failure.

The rest of this chapter is structured as follows. A literature review of previous work done on dynamic preservation of web QoS is given in Section 5.2. This is followed in Section 5.3 by an overview of CBAC. The discount-charge model, which is central to CBAC operation, is described in Section 5.4. The way CBAC uses the system's navigational structure is explained in Section 5.5 by describing CBAC's navigational model. This model is required to get service time estimates which contribute to price calculation in the discount-charge model. CBAC's scheduling of postponed customer requests is considered in Section 5.6. For a system to support CBAC, extra services must be introduced in its application server. Modifications required to the system's application server are considered in Section 5.7. The applicability of CBAC in different Internet Commerce businesses is then discussed in Section 5.8. Finally, in Section 5.9, a summary is given.

5.2 Related Work

Several approaches have been proposed to dynamically enhance QoS, handle system congestion and provide service differentiation in web applications. Some are hardware solutions including web server replication and caching. Such solutions can be expensive financially and they require a software layer for load balancing and consistency. Pure software techniques have also been considered including web server content adaptation and admission control. A common problem of all the solutions mentioned above is that they rely on request rejection during system overload, thus undermining customer satisfaction and the probability of return. This in turn could translate into future profit loss. Another disadvantage

of the above solutions is that they do not provide a comprehensive improvement to all QoS metrics of concern, e.g. response time, throughput, request failure minimisation, system utilisation and system revenue. Rather, one QoS metric is often considered. It is also noticeable that very little work targets specifically Internet Commerce systems and takes into account their special characteristics when studying QoS enhancement.
The rest of this section is structured as follow. The use of server replication to enhance web QoS is discussed in Section 5.2.1. This is followed in Section 5.2.2 by a description of how web caching can improve web server response time. The use of content adaptation methods to improve web QoS is reviewed in Section 5.2.3. Admission control techniques proposed to enhance the QoS of web applications are described in Section 5.2.4. Finally in Section 5.2.5, a discussion is provided on network congestion control and service differentiation methods with an emphasis on pricing techniques.

5.2.1 Server Replication

A widely used hardware solution to reduce the latency of web applications is the use of multiple servers to handle user requests. A variety of server replication architectures exists [64]. The most common architecture is a cluster of servers in a Local Area Network (LAN). Such an arrangement provides a single interface to clients and requires techniques for dispatching client requests among servers in the cluster. Dispatching techniques can be DNS-based (at layer-4 of the ISO network standard) or content-aware (at ISO layer-7). Client-based load balancing techniques have also been considered. Another server replication arrangement uses the *mirrored sites* approach in which web servers are distributed globally on the Internet. The single image of the system is not supported in this setting as the client uses a different Uniform Resource Locator (URL) to access each of the mirrored sites. Different techniques devised for load balancing in replicated servers are reviewed in this section.

A cluster-based web system was first proposed by Katz *et al* [65]. In their work, a cluster of servers is used to ensure the scalability of the National Center for Supercomputing Applications (NCSA) web site. To dispatch HTTP requests among servers, a DNS-based scheduling mechanism is used in which the URL name in a request is mapped to clustered server IPs in a round-robin fashion. However, caching of address mapping at the DNS server and the client can limit the effect of DNS-based scheduling. This is due to the fact that a *Time-to-live (TTL)* pe-

riod is usually assigned to each cached mapping during which DNS scheduling is bypassed. This disturbs the load balancing in the cluster causing some servers to become overloaded. Several solutions have been proposed to overcome this problem. In [66, 67], Colajanni and Yu advocate using adaptive *TTL* algorithms to vary the *TTL* value for each request, based on client request rate and server capacity. Their results show that such algorithms can reduce the imbalance between servers when caching is used.
Cardellini *et al* [68] propose combining DNS scheduling with redirection request mechanisms. Different redirection algorithms are presented in their work including centralised algorithms where the redirection is carried out by the DNS server based on load information collected from the clustered web servers. Distributed redirection is also considered where load information of web servers is maintained by the DNS server and sent to overloaded servers upon request. The DNS server uses a round-robin mechanism to distribute requests among servers. When an overloaded web server is allocated a request, it uses the load information to redirect the request to a less loaded server. Cardellini *et al* also consider redirection of requests for individual clients or for whole domains. Aversa and Bestavros [69] propose another distributed redirection mechanism where the clustered web servers exchange their load information without the need for a central component. When the number of connections a web server is handling exceeds a threshold, the web server starts redirecting request packets to other servers in the cluster based on the load information it maintains.

Content-aware distribution is another method used to balance the load in a web server cluster. Yang and Luo [70] and Zhang *et al* [71] propose a content placement scheme partitioning the content of a web site in different web servers. A front-end server (*distributor*), interacting with clients, holds information on the partitioning of content among back-end web servers. It distributes client requests to the appropriate back-end web server based on the content-related information it holds. In [72], Luo and Yang extend their content-aware request scheduling with request redirection in the event of web server overload or failure. To minimise the overhead of redirection, Cherkasova and Karlsson [73] propose taking workload properties into account where files for popular web pages are contained in each web server in the cluster and thus can be served by any of the servers minimising redirection. Less popular pages are partitioned among all the servers. Zhu *et al* [74] present a master/slave web cluster architecture in which static pages are processed locally by the master while dynamic pages are redirected to slaves to distribute the load. A more dynamic replication of content is advocated by Pierre *et al* [75] in which replication strategies are applied dynamically and on an individual basis to each document in the web site.

Research has also been done on combining both DNS and content-aware scheduling. Aron *et al* [76] apply the content-aware request scheduling mechanism to a set of back-end web servers in a cluster. A front node uses DNS scheduling to forward requests to back-end servers. If a web server does not have the requested content, it redirects the request to the web server that does. Dias *et al* [77] suggest a web cluster architecture consisting of a set of front-end nodes and a set of back-end nodes connected by a switch. DNS scheduling is used for load balancing at the front-end nodes while the distribution of web site content at the back-end nodes is used to balance their load. Chen and Iyengar [78] present a similar two-tier web cluster but incorporate client information and server load in the scheduling policy of the front-end nodes. Different versions of the web site content varying in quality (e.g. text-only, minimum graphics, etc.), are distributed at the back-end nodes.

Client-based approaches to distribute load among web servers in a cluster have also been proposed. Casalicchio and Colajanni [79] and Ciardo *et al* [80] advocate the use of client request size and its impact on web server load to distribute requests. The distance between the client and the web servers is examined by Sayal *et al* [81] and Shaikh *et al* [82] as the method for load balancing. Delegating load balancing to clients is an approach that has also been considered. Yoshikawa *et al* [83] and Vingralek *et al* [84] use software agents residing on the client to choose the web server that will provide the best response time. The use of a client proxy for this task is proposed by Fei *et al* [85].

Other solutions have been developed for replicated servers which are distributed geographically across a wide area network. These include network bandwidth probing presented by Carter and Crovella [86] which focuses on examining the bandwidth and congestion along paths to alternative web servers. Yu and Lin [87] propose using a QoS broker to aid a client in negotiating with web servers for QoS guarantees. Environments have also been developed to support distributed web servers in terms of load balancing strategies and content replication [88] [89] [90].

5.2.2 Caching

Web caching is a well-established approach to reduce the response time of web servers. Research has found that a caching algorithm usually achieves a maximum hit rate of 50% [91]. A cache miss would cause the web server response time to increase due to the extra unbenefical cache processing. In this section, techniques to increase web cache hit rates are discussed including cache prefetching, replacement policies and cooperative caching. Dynamic content caching is also considered.

One way to increase a web cache hit rate is to predict the documents that are likely to be requested and load them in the cache. Kroeger and Long [92] have found that web caching combined with prefetching reduces latency by 60%. This indicates the effectiveness of prefetching in increasing hit rates and thus reducing the number of requests that are handled on the server. Padmanabhan and Mogul [27] propose prefetching directly from a web server to a client. Their work is based on a prediction algorithm which constructs a dependency graph of access patterns to different files stored at the web server. A similar speculative framework which is based on analysing accessed documents' dependencies has been presented by Bestavros [28].
Work has also been done on prefetching between proxies and web servers. Markatos and Chronaki [93] propose a prefetching technique in which a web server regularly pushes its most popular documents to proxies. Those proxies can then forward such documents to their clients. Chinen and Yamaguchi [94] advocate the use of interactive prefetching in which a requested HTML document is parsed to gather its references to other resources (HTML pages, images, etc). The referenced resources are then prefetched to reduce latency of subsequent requests.
Other work presented by Fan *et al* [95] considers prefetching between a browser client and a proxy. Their technique takes advantage of the idle time between user requests to a proxy in order to predict which cached documents are likely to be requested. Those documents are then pushed to the user.

Another approach to maximise web cache hit rates is the use of effective web-specific cache replacement policies. Traditional replacement policies exist including *Least Recently Used (LRU)* and *Least Frequently Used (LFU)*. However, work has been done on replacement policies which take cost into account when deciding which documents should be replaced. Cao and Irani [96] has devised *GreedyDual-Size*, a cache replacement algorithm which uses document size and latency cost of loading a document into the cache as the criteria for replacement. *LNC-R-W3* is another algorithm proposed by Scheuermann *et al* [97] which keeps the documents requiring the longest delays to load into the cache. *LNC-R-W3-U* is an extension to *LNC-R-W3*, devised by Shim *et al* [98], which additionally considers validation of documents based on a time-to-live method. Document fetching delay is also used in a replacement technique presented by Wooster and Abrams [99] in addition to considering documents that are retrieved over low bandwidth links. Niclausse *et al* [100] consider a replacement method that combines network latency, documents' size, their access frequencies and time elapsed since their last updates. A different cost-aware technique proposed by Tewari *et al* [101] considers cache resource usage cost when replacing documents in the cache.

Cooperative caching is another approach that can be used to reduce web latency. In this approach, a group of cache proxies cooperate to share cached objects. Work has been done on devising techniques to facilitate this cooperation. Fan *et al* [102] have devised *Summary Cache*, a protocol in which each proxy keeps a summary of URLs cached at each of the cooperating proxies. On a cache miss, the proxy checks those summaries for potential hits before sending queries to other proxies. Rabinovich *et al* [103] present a technique in which a proxy experiencing a cache miss uses a *neighbourhood graph* and a *communication graph* to query proxies with the shortest distance. Another technique based on finding the nearest cache is presented by Zhang *et al* [104]. The authors propose organising web servers and cache servers into clusters. Multicasting is used locally in a cluster and across clusters to query a document. Song *et al* [105] present a web server accelerator based on cooperative caching. Cache memory is distributed across the accelerator nodes. Content-based and TCP-based routing are studied as possible methods to find cached documents in the accelerator.

All the work discussed above focuses on static content. However, caching of dynamic web content, which is typically part of an Internet Commerce system, can have a greater effect in reducing the system's response time. This is because dynamic content needs substantially more CPU time to be constructed. However, a problem of caching dynamic content is ensuring cache consistency as dynamic documents are updated. Holmedahl *et al* [106] implement a weak consistency protocol for cached dynamic content which allows access to old cache content for a period of time. The system administrator sets a *TTL* field for different documents causing the cached document to expire after a period of time. This lazy invalidation technique is convenient for web sites with infrequent updates of documents. Zhu and Yang [107] propose a technique in which web pages are grouped into classes based on page URLs and client information. Classes are invalidated on a group basis using lazy validation. The authors also consider using page precomputation to prefetch popular dynamic pages when the server load is low.

Dynamic content fragment caching has also been considered. Challenger *et al* [108] propose *Data Update Propagation* which maintains data dependency information between cached objects and underlying data that frequently change and affect cached objects' content (object fragments). On a change in the underlying data, the system queries the dependency information to determine which objects are affected. This contributes to making the decision of which documents should be invalidated or updated. Ramaswamy *et al* [109] propose an automatic fragmentation mechanism which detects fragments that are shared among different dynamic pages and have long lifetime characteristics. Generated fragments are considered as the best candidates for caching. Datta *et al* [110] exploit the fact that dynamic pages have a content and layout dynamic properties. Based on that, the authors propose caching dynamic content fragments in proxy caches but querying the source site for dynamic document layout. The proxy fills the cached content of the document as specified by the layout.

Active caching is another technique used for caching dynamic content. In this scheme, a proxy cache can service a request by executing a query over the content of the cache. Cao *et al* [111] propose *Active Cache* in which cached documents at a proxy are attached to applets provided by the web server. On a cache hit, a proxy executes the cache applet over the cached document it is associated with and the result is returned to the client. A similar technique is presented by Meira Jr. *et al* [112]. Luo and Naughton [113] propose a form-based active caching technique that uses parameterised query definitions. Such query definitions are instantiated with user requests' parameter values. Yuan *et al* [114] has found that active caching can reduce latency by two to three folds.

5.2.3 Content Adaptation

Researchers have examined web content adaptation as a method to provide QoS and service differentiation on the Web. Most work done on this approach is directed towards optimising response time and system utilisation. Different methods have been considered ranging from statically providing different versions of a web site to online adaptation. Content adaptation of dynamically generated web pages has also been considered.

As a static method for content adaptation, Abdelzaher and Bhatti [115] propose having different content trees for a web site ranging from full to minimum content. The selection of the content tree from which a request is served depends on the current web server utilisation. A similar approach is presented by Muntean *et al* [116] in which several versions of a web site are provided. A class of users is assigned to each web site version based on their performance requirements. Krishnamurthy *et al* [117] classify users according to their connection capabilities. Given different content variations, users with poorer connections are provided with a reduced content version.

Mechanisms for online adaptation of web page content have also been proposed. Chandra *et al* [118] present a transcoding technique to vary the size of multimedia web objects dynamically. In their work, a bandwidth is assigned to each group of users and the transcoding done is proportional to that bandwidth. Wills *et al* [119] focus on the packaging of embedded objects making web pages into *bundles*. They found that reducing the number of connections required to fetch web objects individually reduces response time, network and server load. Work has also been done on the use of compression to reduce the size of web pages in order to provide better response time. Fox and Brewer [120] advocate a distillation policy that exploits specific properties of datatypes in web pages. For instance, an image is compressed by reducing its size and/or colour depth. Mogual *et al* [121] apply delta-encoding and data compression to HTTP responses to reduce response size and delay.

Other work has been done on the adaptation of dynamically generated web pages. Mohapatra and Chen [122] describe *WebGraph*, a framework in which a graph is managed for each dynamically generated web page. The graph consists of *weblets* which represent static and dynamic objects in the web page. Using this graph, only the parts of the page that have changed (dynamic objects) are recreated when a request is made, thus reducing the latency. Ramaswamy *et al* [109] consider the automatic fragmentation of dynamic web pages to benefit content generation. They propose algorithms for detecting fragments that are shared among different dynamic pages and have long lifetime characteristics, thus reducing the delays required for content generation. Abdelzaher and Bhatti [115] propose using scripts with varying complexity in their content trees approach described above.

Optimising the number of dynamically generated web pages needed to fulfil the user's required service can reduce service time and system congestion. An example of such approach is Amazon.com's *one-click ordering* patent granted in 1999 [123]. It is an alternative method to the typical shopping cart ordering mechanism in Internet Commerce systems. If *one-click ordering* is enabled and a client requests purchasing an item, client information (possibly maintained using cookies), the required item information and an ordering button are bundled into a web page which is sent to the client. One click by the client on the ordering button causes the order to be placed.

5.2.4 Admission Control

Different admission control strategies have been proposed to enhance QoS in web applications. Some adopt a feedback control approach while others are based on avoiding handling requests that will not be completed successfully. Research on admission control techniques that provide differentiated service for a set of user classes is well-established. Request processing requirements is another criterion considered as the basis for admission control. However, all admission control policies proposed rely on request rejection as a way to reduce system load in the state of system overload. Studies tend to consider individual QoS metrics such as response time, throughput, system utilisation or system revenue without taking a comprehensive approach to web QoS enhancement. Most researchers focus on general web servers performance with some work on web hosting and multimedia and very little work targets Internet Commerce specifically. In this section, work done on applying admission control to different web applications is reviewed.

Work has been done on the use of feedback control theory as the basis for admission control in web sites to provide QoS guarantees. Abdelzaher and Shin [124] consider modelling a web server as a classical feedback control problem. A software *sensor* is used to monitor the server's CPU utilisation. Measurements obtained by the *sensor* are used to find optimal feedback controller parameters. Admission control is implemented in an *actuator* which, based on controller parameter values, reduces the current load of the system to avoid server load. The reduction of load is achieved by dropping current requests or providing degraded service for some requests. The authors provide a time-varying linear model of a web server. In [125], Robertsson *et al* model a web server as an M/G/1 queue. The length of the queue is considered as a measure of system load. A controller is used to reject requests when a maximum queue length is reached.

Multiple-Input Multiple-Output (MIMO) control theory has also been applied to web server QoS admission control. This allows more than one control parameter to be used in the feedback loop. Diao *et al* [126] consider a request rejection and dropping admission control mechanism based on *MIMO* control with CPU and memory utilisation as the control parameters. The authors argue that their approach is more accurate (than single parameter feedback) as it captures the interaction between the two parameters.
The feedback control approach reacts to performance degradation after it already occurs. Combining system load prediction with feedback control to avoid server performance degradation is a problem some researchers have focused on. Abdelzaher *et al* [127] [128] [129] combine queueing theory with feedback control to predict the resource allocation required to meet response time requirements for user requests. Feedback control combined with queueing is also used by Menasce *et al* [130] to predict the optimal Internet Commerce system configuration required to meet desired QoS levels. Request rejection is one of QoS metrics considered in addition to response time and throughput.

Another web server admission control approach is based on avoiding serving user requests that are likely to fail eventually. Using this approach, Iyer *et al* [131] propose two schemes to handle web server overload. The first is to use an intelligent Network Interface Card (NIC) to selectively drop packet requests when the system is overloaded. The other scheme is to use *traffic throttling* in which the overloaded server sends feedback messages to the request source requiring a reduction in the rate requests are sent. Another technique studied by Cherkasova and Phaal [132] uses a session-based admission control mechanism in which a user request is accepted only if system capacity can handle all subsequent requests in the user session. A simple rejection policy is used in which a request (and the whole session) is rejected if a maximum number of connections is reached or if the user tries to resend a request a certain number of times. A similar admission control approach is proposed by Chen and Mohapatra [133].

Extensive work has been done on admission control techniques based on providing differentiated service for multiple user classes. Most of the work has focused on admission control for general web applications rather than specifically targeting Internet Commerce systems. Bhatti and Friedrich [134] present a web server admission control policy in which requests are classified according to the class of the user making the request or the content required to fulfil the request. Given

the request classification, a request can be rejected or scheduled according to the class priority. Voigt *et al* [135] propose three techniques to prioritise user requests. The first technique is similar to Bhatti and Friedrich's work and gives priorities to requests based on the content required. The second technique proposed by Voigt *et al* drops user request packets based on the client's IP address. Jamjoom *et al* [136] also suggest associating requests with user classes at the packet level using a set of rules specified by the system administrator. Voigt *et al*'s third technique uses a listen queue in which requests are re-ordered according to their priority of service. A similar queueing technique is presented by Chen and Mohapatra [137]. Vasiliou and Lutfiyya [138] schedule requests for different user classes separately and specify a limit on the number of requests accepted for each class. Li and Jamin [139] present a bandwidth-based admission control technique which allocates server bandwidth to user classes. A user request is rejected if the user's allocated bandwidth is exceeded or the server is fully utilised.
Some researchers have considered a more flexible differentiated service admission control for web servers which controls the bounds between different user classes handling. The purpose of this is to avoid request starvation. Lee *et al* [140] present a proportional-delay admission control technique in which relative time delays are guaranteed for different user classes. Kanodia and Knightly [141] have devised a technique which provides user class isolation, with a fraction of system resources allocated to each user class. The technique proposed by the authors allows a class to utilise unused resources of other classes. Chen *et al* [142] present a technique in which user classes are dynamically allocated system resources by estimating the request rate of tasks.

Shortest job first admission control has also been suggested. Crovella *et al* [143] use connection length as the basis for admission control. Connections with shortest duration are given highest priority. Determination of connection length is based on the size of the document requested (only static documents are considered). A Shortest Remaining Processing Time First admission control is proposed by Schroeder and Harchol-Batler [144]. Cherkasova [145] proposes a tunable admission control strategy which can be adjusted between FIFO and shortest request first.

Other researchers advocate applying admission control at different stages of a service offered by a web server. Welsh *et al* [146] [147] consider a service as a composition of stages. Requests are dropped at each stage to eliminate performance bottlenecks. Carlstrom and Rom [148] present a similar approach in which sessions are divided into stages and requests are queued for each stage. By controlling the resources allocated to each stage in the session, the authors focus on maximising the profit of an e-commerce system.

The use of admission control for hosting services has also been considered. Verma and Ghosal [149] propose an admission control technique to maximise profit of a hosting service. Given a set of *Service-Level Agreements (SLAs)* and finite server capacity, a subset of user requests are accepted so that profit is maximised. A *SLA* contains reward and penalty parameters incurred by the system. A *Shortest Remaining Job First (SRJF)* admission control policy is used to prioritise requests. Abdelzaher *et al* [150] [151] use the concept of *QoS contract* between a web service provider and a user. Admission Control is applied at two levels: when new contracts are accepted in addition to a request-level admission control in which requests with a low priority contract are rejected in an overloaded situation.

Admission Control techniques to enhance QoS have also been considered for multimedia servers. Chen and Li [152] propose an admission control strategy to maximise the reward received by a multimedia web application. Vin *et al* [153] propose a technique that exploits the variation in access times to media blocks in disk and playback variations to improve utilisation of system resources and provide QoS guarantees to individual users. Lee and Sabata [154] have devised an algorithm which is based on setting aside a fraction of system resources as reserved in a multimedia application. When the system is about to enter a congested state, the reserved system resources are used intelligently by distributing small amounts of resources to user requests in order to maximise system benefit. Adding admission control to a cluster of web servers has been considered by Zhu *et al* [155]. In their work, the authors propose periodically repartitioning the cluster in response to workload variations in order to provide differentiated service to a set of user classes.

Little work has focused on applying admission control specifically to Internet Commerce systems. Menasce *et al* [156] [157] describe a priority-based resource management policy to maximise profit. Customers navigating the e-commerce site are classified based on session length and the accumulated money in their shopping cart. A customer navigating the site for too long with not much value in their shopping cart is given a low priority in terms of processor and disk use. Another recent work presented by Elnikety *et al* [158] proposes *Shortest Job First* admission control policy to reduce response time and sustain throughput in an Internet Commerce system. Requests are rejected when a capacity threshold is reached. Job execution costs are learned online.

5.2.5 Network QoS and Pricing

Research on congestion control, QoS assurance and service differentiation in networks is well-established. Standards such as *DiffServ* [159] enrich the IP protocol with a framework for enabling QoS-based service differentiation over the Internet. TCP modifications to achieve congestion control and service differentiation are considered [160] [161] [162]. Measurement-based techniques to control traffic congestion in networks have been studied [163] [164] [165] [166] [167]. Dynamic pricing is another approach considered where usage price is varied to control congestion in a network. Exploiting price as a tool to control a congested resource has influenced the development of CBAC which applies dynamic pricing in Internet Commerce QoS enhancement. In the rest of this section, work done on the use of dynamic pricing to control network congestion is reviewed.

A study presented by Cocchi *et al* [168] concluded that a set of variable prices exist that make every user more satisfied with the combined cost and performance of a network. Mackie-Mason and Varian [169] [170] [171] describe an economic analysis for pricing a congested network. They propose charging a congested-switch-user a price reflecting the social cost that they will impose on other users, i.e. the cost of preventing or slowing down other users' packets going through the switch. Mackie-Mason and Varian describe an implementation of their approach, *the smart market*, where each packet would have a *bid* field in its header to indicate how much its sender is willing to pay. The network would admit all packets whose bids exceed the current network threshold. Gibbens and Kelly [172] show that the *smart market* approach can be realised by using a decentralised mechanism where some packets in the network are marked indicating congestion. Charging the end-user for each marked packet will motivate efficient use of the network.
Shenker *et al* [173], through their *edge pricing* mechanism, propose determining usage prices at network end-points by estimating the congestion along the expected path of a packet between source and destination. This eliminates the need for the distributed calculation of the price along the path. Wang and Schulzrinne

[174] describe an approach in which congestion charge is calculated using an iterative *tâtonnement* process [175], an economic formula that determines the current price based on demand, supply and the previous price. Odlyzko [176] presents a simpler network pricing framework, *Paris Metro Pricing.* It is based on dividing the network into separate logical channels and assigning a usage price for each of those channels. Differentiated service is achieved as channels with higher prices attract less traffic and thus provide better QoS. Leyton-Brown *et al* [177] use probabilistic price variations to distribute network load across different time slots.

5.3 An Overview of CBAC

CBAC is a novel approach to preserve QoS in Internet Commerce systems. It is based on controlling system congestion by encouraging customers to postpone their requests when the system is approaching overload. When the system is close to a congested state, accepting more customer requests will make the situation deteriorate. However, refusing requests will have a negative effect on customer satisfaction. A solution is to postpone requests by encouraging customers to leave the system at a time of likely congestion and come back at a later, less-loaded, time. Dynamic pricing could be used to drive such customer encouragement. CBAC follows the strategy explained above. It is an admission control technique that manages congestion in Internet Commerce systems without rejecting customer requests. The novel technique is based on a discount-charge pricing model to encourage customers to postpone their requests in a highly loaded situation. Using this model, a customer is offered a discount on the services they are likely to request in return for postponing such service requests until a less-loaded time interval. An extra charge is imposed on customer requests if the customer decides to go ahead with the requests during the high load period. CBAC's discount-charge model is sensitive to the following:

- The current system load when the service request is made.
- An estimation of service time to fulfil the customer's request and all subsequent requests dependent on it.
- Service time estimation for the postponed requests already scheduled for execution at the time the service request is made.

In order to estimate the service time of an incoming customer request and all its subsequent requests in a dependent Internet Commerce service model, CBAC includes a navigational model of the system it is applied to. This model provides a framework for calculating the required time estimates. CBAC also includes a scheduling mechanism with load forecasting which is used to schedule postponed user requests in more lightly loaded time periods.

5.4 Discount-Charge Pricing Model

As described in Section 5.3, CBAC exploits service price variation in order to encourage customers to postpone their requests to avoid Internet Commerce system overload. A description of CBAC's discount-charge approach for price variation is given in this section. The factors affecting the discount-charge model are first described in Section 5.4.1. This is followed in Section 5.4.2 by a formal definition of the model.

5.4.1 Factors Affecting the Model

When the system's used resources become equal or greater than a specified threshold ϵ, CBAC is applied to every request received by the system. The reason behind basing CBAC admission on user request-level rather than session-level is threefold. Firstly, a faster and more direct reaction to changes in system congestion is achieved when reducing CBAC's admission granularity. Basing CBAC admission on the cumulative load of user sessions will cause a slower reaction in triggering CBAC on and off. Secondly, using a user session admission policy will introduce discrimination among users. This is because a selection criteria will then need to be used to choose the user sessions CBAC should be applied to in order to reduce system congestion. Finally, monitoring and maintaining the cumulative load of each user session would be expensive in terms of system resource usage.

The amount of discount/charge, A, offered to the customer when CBAC is applied is dependent on the following factors:

- The percentile of available system resources R, when $1 - R \geq \epsilon$. A negative coefficient should govern the relation between A and R. As the amount of available resources decreases, the discount or charge offered should increase encouraging a customer more strongly to leave the system at the time of the request.

- The requested service estimated service time D. This includes the service time of the current request and all subsequent requests the customer is likely to request in a session. Estimation of service times can be obtained by monitoring the system's service times at low load when the system is in a non-congested state. A has a directly proportional relationship with D. Customers requiring more system resources to execute their request should be encouraged more strongly to leave the system. Estimation of service time using CBAC's navigational model is described in Section 5.5.

- Finally, the service demand, S, for jobs already scheduled for execution in the time interval a customer request is received. A has a directly proportional relationship with S. Customers should be encouraged more strongly to leave the system if bigger jobs are scheduled to be executed at the time the customer request is made. CBAC's postponed request scheduling is described in Section 5.6.

Factors that provide user differentiation are not included in CBAC's pricing model. CBAC is designed to provide a fair admission control policy among users and thus factors such as the financial value in a user's interactions with the system are not considered in CBAC's pricing model. Only the effect of the user interactions on the system's congested state is taken into account.

5.4.2 Formal Definition of the Model

Formally, A can be defined as:

$$A = f(R, D, S)$$

$$A = \left\{ \begin{array}{ll} \min[|\beta D + \delta S - \alpha R|, A_{max}] & (1 - R \geq \epsilon) \\ 0 & (0 < 1 - R < \epsilon) \end{array} \right\}$$

where β, δ, $\alpha \in \mathbb{R}$ and $0 < \beta$, δ, $\alpha \leq 1$. β, δ, α are weights chosen to reflect the required contribution of D, S and R respectively in determining A. A_{max} is an upper bound for A. ϵ is the used resources threshold. R represents the spare capacity of the system bottleneck:

$$R = 100 - max[R_0, R_1, \ldots, R_n]$$

where R_i is the utilisation of the i^{th} system resource in percentile.

In the definition above, it is assumed that the amount of discount is always equal to the charge offered to the customer. A charging factor C could be introduced to the model in order to vary the amount of discount and charge offered:

$$A_{ch} = CA$$

$$A_{dis} = A$$

where A_{ch} is the amount of charge offered and A_{dis} is the amount of discount. C values can be chosen to increase the charge imposed on a customer requiring an immediate execution of a service request when CBAC is applied. Thus, the urgency of customer requests is exploited to increase system profit.

Throughout this thesis, the linear function above is used to calculate A. However, other types of functions could also be used such as exponential functions. As described in Section 7.2, a future work direction is to make the discount/charge function adaptable, not only to system load, but also to profit levels and market demand-supply.

5.5 Internet Commerce Service Modelling

A common feature of Internet Commerce systems is offering dependent services. A user usually interacts with an Internet Commerce system through a sequence of dependent service requests forming a *user session*. After a service request in a *user session* is fulfilled, the user is usually given the option to request from a set of further services. The decision the user makes of which next service to request is dependent on the outcome of the service requested previously.
Determining the service time for a given request in such a dependent service model would require estimations of the service time of that request and also the service times of all the services the user is likely to request in the remainder of the session. The total estimated service time of such services contribute to CBAC's discount-charge model as described in Section 5.4. In order to derive such estimates, CBAC includes a navigational model of the system to which it is applied. This model is used as a framework to analyse the navigational dependencies in the system. The model includes a formal method for service time estimation given the current user request.

This section is structured as follows. In Section 5.5.1, an analysis of service dependencies in TPC-W's navigational pattern is given. This is followed in Section 5.5.2 by a description of a sample navigational structure derived from TPC-W's navigational pattern. CBAC's navigational model is described in Section 5.5.3. Finally, the use of the model in service time estimation is formalised in Section 5.5.4.

5.5.1 TPC-W, a Dependent Internet Commerce Service Model Example

The TPC-W standard [3] is used to study the navigational structure of a typical dependent Internet Commerce service model. As described in Section 3.2, TPC-W's service model contains 14 dependent services with 3 alternative navigational patterns for browsing, shopping and ordering scenarios. The navigation between different services is governed by the threshold mechanism described in Section 3.2.1. To ease the understanding of dependencies between services, TPC-W's threshold values can be converted to probabilities of navigation between different services as follows:

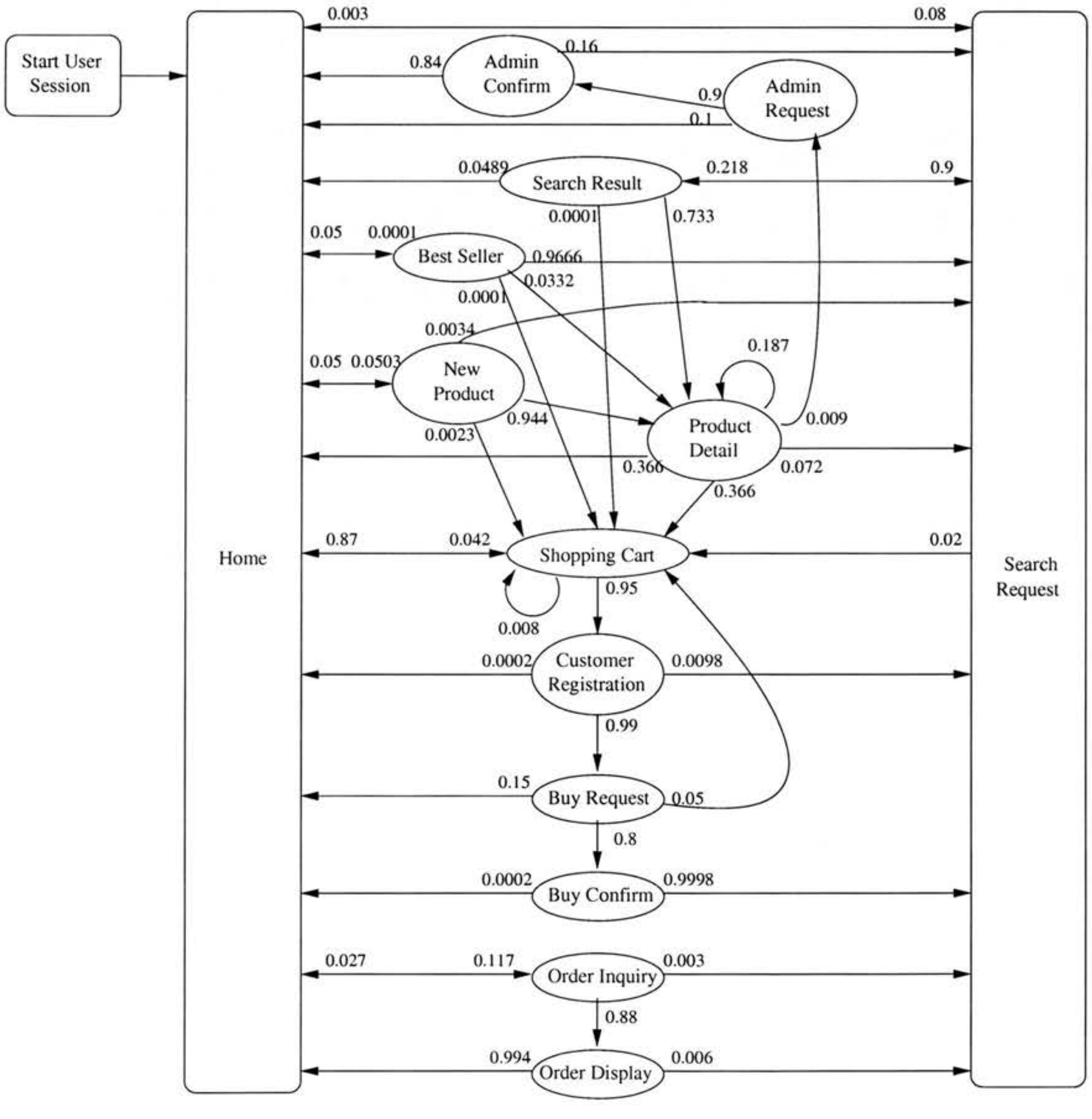


Figure 5.1: TPC-W's Service Model Navigational Probabilities in an Ordering Interval (derived from [3])

Given that the thresholds for navigating from service s_c to services s_0, $s_1,\ldots$, s_i are t_0, $t_1,\ldots$, t_i respectively, where $t_0 < t_1 <\ldots< t_i$, the probability of navigating to service s_i from s_c

$$P_{ci} = \tfrac{t_i - t_{i-1}}{9999} \quad (i > 0)$$

and the probability of navigating from s_c to s_0 is

$$P_{c0} = \frac{t_0}{9999}$$

That is, the probability of navigating to a service can be calculated by subtracting the preceding service threshold from its threshold and dividing by 9999. Figure 5.1 shows the probabilities of navigation between TPC-W services in an ordering scenario where 50% of the interactions are browsing requests and 50% are ordering. TPC-W's ordering scenario threshold values are included in Appendix C.

5.5.2 Navigational Structure Sample

TPC-W's navigational structure for an ordering scenario, shown in Figure 5.1, has been modified and used as an input to CBAC's navigational model which will be described in Section 5.5.3. The modifications made are designed to simplify TPC-W's navigational structure while preserving its realistic characteristic in terms of functionality and user navigation. Figure 5.2 illustrates the modifications which are outlined below:

- In order to clearly mark the beginning of a session, it is assumed that the user's first service request is always *Home*.

- The navigational structure is simplified so that the minimum number of services are requested to place an order. Thus, services that are not required as intermediate steps toward making an order are removed (and the interactions they are involved in) including: *Order Inquiry*, *Order Display*, *Admin Request* and *Admin Confirm*.

- Similarly, navigations that do not directly lead to making an order are removed including navigations: *Search Result* $\rightarrow$ *Search Request*, *Product Detail* $\rightarrow$ *Search Request*, *Best Seller* $\rightarrow$ *Search Request*, *New Product* $\rightarrow$ *Search Request*, *Product Detail* $\rightarrow$ *Product Detail* and *Customer Registration* $\rightarrow$ *Search Request*. All navigations going back to *Home* from other services are also removed.

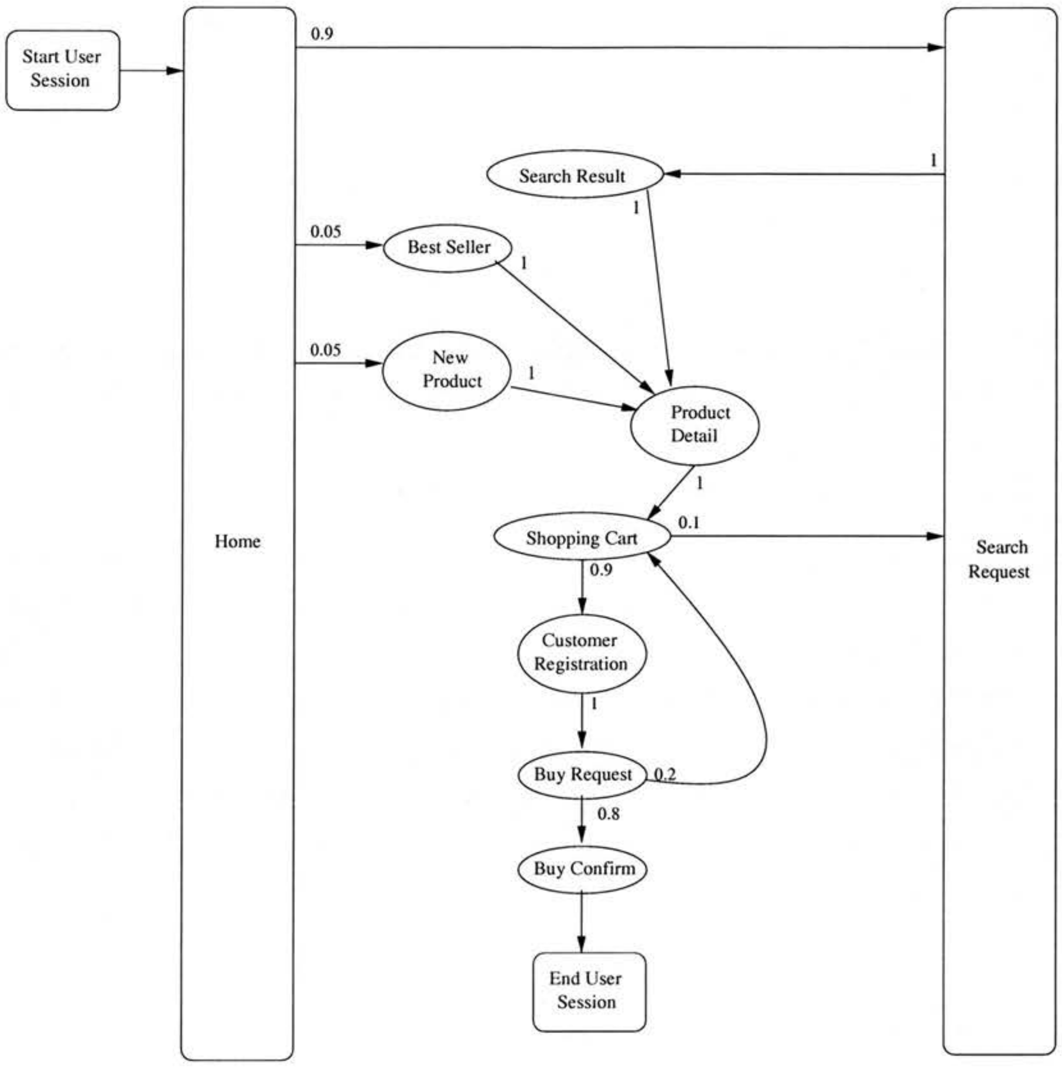


Figure 5.2: Internet Commerce Navigational Structure Sample

- *Shopping Cart* is a central service in TPC-W's navigational structure as it is involved in interactions with many other services. In order to simplify the navigational structure, some of those interactions are removed while keeping the two main objectives of a *Shopping Cart* service feasible, which are: adding as many items as required to the cart and allowing the customer to start the ordering process and go back to *Shopping Cart* if deciding to make changes in the items to be purchased. The following navigations involving *shopping Cart* are removed: *Home* → *Shopping Cart*, *New Product* → *Shopping Cart*, *Best Seller* → *Shopping Cart*, *Search Result* → *Shopping Cart* and *Shopping Cart* → *Shopping Cart*. The *Search Request* → *Shopping Cart* navigation is replaced by *Shopping Cart* → *Search Request*.

- It is assumed that a session always ends with a customer making an order of purchase. This simplifies the profitability analysis of the system. Thus, TPC-W's navigational structure is modified so that *Buy Confirm* is always the last request in a session. The navigation *Buy confirm* $\rightarrow$ *Search Request* is removed. Requesting a *Home* service after a *Buy Confirm* should start a new session.

The probabilities of navigation have been modified to reflect the changes in TPC-W's navigational structure. The new probabilities are illustrated in Figure 5.2.

5.5.3 CBAC's Navigational Model

As described in Section 5.4, when CBAC is applied in a highly loaded system, the estimated service time of a service request, and all subsequent requests in the user session, contribute to determining the discount or charge the user is offered for service fulfilment. To determine such an estimate in a dependent Internet Commerce service model, an analysis of the navigational structure of the system is required. This is achieved by constructing a CBAC Navigational Model (CBAC-NM) of the system which provides a framework for calculating service time estimates. The input to CBAC-NM is the system's navigational structure including services, how they are interconnected and the navigational probabilities between them. Estimates of service times of individual services are also required. In CBAC-NM, services are categorised into two types: atomic and composed. An atomic service is a service that a user can request in one interaction, whereas a composed service consists of a cycle of atomic and composed services that are requested in more than one interaction. atomic and composed service properties include:

- Every individual service in the model is an atomic service.
- An atomic service can be part of a composed service but not vice versa.
- An atomic service can be directly part of one composed service only.
- An atomic service cannot be part of another atomic service.
- A composed service can be part of another composed service.
- A composed service cannot be part of more than one composed service.

The point of distinguishing composed services is to capture the pattern of interaction where the user can repeatedly request a service loop consisting of a set of

services. As described in Section 5.5.4, this representation is used in CBAC-NM to obtain time estimates for a service loop given the service time of the services making up the composed service and the number of times the composed service is requested (by requesting the services making it up).
Below is a summary of the steps required to construct a CBAC-NM of the CBAC-enabled system:

1. Identify atomic services in the system.

2. Determine composed services by identifying loops in the system's navigational structure.

3. Identify links between services and probabilities of navigation through those links.

4. Estimate the service time required for each of the atomic services in the model.

A graphical representation of CBAC-NM has been devised. Figure 5.3 illustrates the representation's entities. An atomic service is represented by an ellipse con-

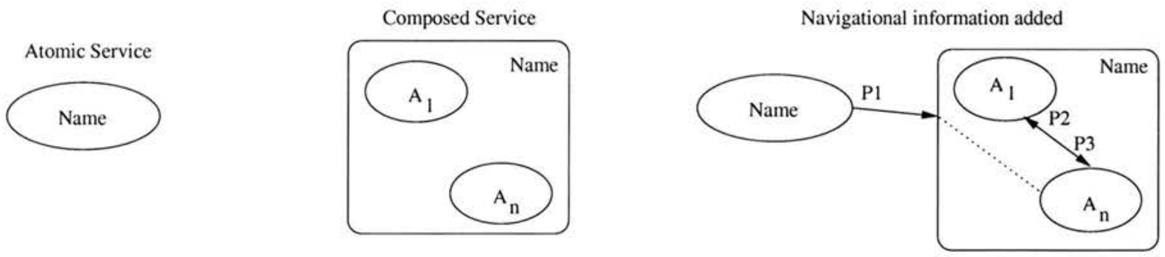


Figure 5.3: CBAC-NM Graphical Representation Entities

taining the service's name. A rectangle represents a composed service with its name in the rectangle's top right corner. The atomic services composing a composed service are included in the rectangle. An arrow linking two services denotes a possible navigation in the arrow's direction with the probability of navigation shown on the arrow. A dashed line extending a navigational arrow is used to illustrate which atomic services of a composed service interact with external services.

A CBAC-NM has been derived for the sample navigational structure described in Section 5.5.2 and shown in Figure 5.2. The graphical representation of the model is depicted in Figure 5.4. The following atomic services have been identified:

Home, Best Seller, New Product, Search Request, Search Result, Product Detail, Shopping Cart, Customer Registration, Buy Request and Buy Confirm. Two composed services are denoted C1 and C2, corresponding to the two loops identified in Figure 5.4:

- *C1: Search Request → Search Result → Product Detail → Shopping Cart → Search Request.*
- *C2: C1 → Customer Registration → Buy Request → C1.*

Navigational links between services and probabilities of navigations in the input navigational structure are mapped to CBAC-NM. In and out external navigations for composed services are assigned the probabilities of their atomic services involved in such navigations. Probabilities of navigation should be obtained by analysing the history of user access patterns. For CBAC-NM to be complete, service time estimates are required for each of the atomic services identified. Such estimates should be determined off-line by performing experiments to determine the service time of each of the atomic services at low load.

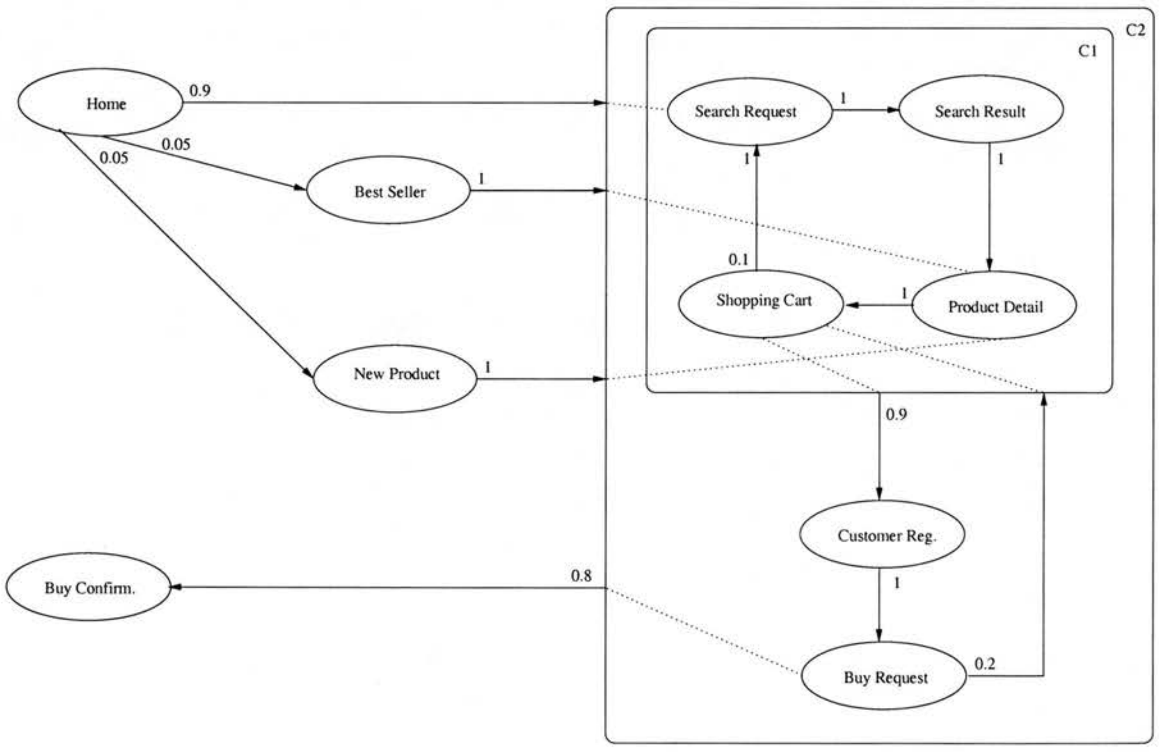


Figure 5.4: CBAC-NM for Sample Navigational Structure Shown in Figure 5.2

5.5.4 Service Time Estimation

As described in Section 5.5.3, using CBAC-NM, service time estimation of a customer request and its possible subsequent requests can be determined. To obtain such an estimate, CBAC-NM uses the recursive definition described below.

We assume a set of services s_0, s_1,..., s_m with estimated service times t_0, t_1,..., t_m. Each service s_i has a list of next services n_0,..., n_j with request probabilities P_{i0},..., P_{ij}. If the last unfulfilled service request the user made is s_r, then the total estimated service time of the remaining services the system is likely to fulfil for the user T_r is:

$$T_r = t_r + \sum_{k=0}^{j} P_{rk} * T_k$$

The service time of an atomic service s_i is just t_i. However, if s_i is a composed service with a cycle of atomic and composed services a_0,..., a_z, its estimated service time would be:

$$t_i = L_i \sum_{k=0}^{z} t_k$$

where L_i is the number of times the user is likely to request the composed service s_i and t_0, t_1,..., t_z are the service times corresponding to a_0,..., a_z. One way to determine the value of L_i is to use information acquired from the user, when starting a session, about the nature of the task required. Users could be asked about the number of items they are likely to purchase. This provides a good estimate for L_i as a user navigating an Internet Commerce site repetitively requests the same group of services for each item they purchase, for example, requesting *C1* shown in Figure 5.4. Another approach to determining L_i could be based on analysing the system's access pattern logs to determine the average number of times users requested each of the composed services in the navigational model.

5.6 Customer Postponed Request Scheduling

As described in Section 5.3, scheduling of a service request is required when the requesting customer accepts a discount to postpone their service request. In this section, CBAC's scheduling mechanism is explained. The scheduling problem is formalised in Section 5.6.1. This is followed in Section 5.6.2 by a discussion of the alternative scheduling strategies that could be used in CBAC. Then in Section 5.6.3, a description is given of how system load forecast is used in CBAC's scheduling. Finally, the scheduling contribution to the outcome of the discount-charge pricing model, described in Section 5.4, is formalised in Section 5.6.4.

5.6.1 The Scheduling Problem

Suppose that time is split into discrete equal intervals I_0, I_1, I_2,..., I_n. Each interval has duration σ where n and $\sigma \in \mathbb{N}$. Let j_n be the set of service request jobs scheduled to be executed in I_n. Also, let $C(j_n)$ be the system processing capacity required to execute j_n. A new service request for job j_{new} can be scheduled to be executed in I_n if

$$C(j_n) + C(j_{new}) + C(l_n) \leq u * \sigma$$

where l_n represents a forecast of possible incoming, not scheduled, service requests in I_n (see Section 5.6.3). $C(j_{new})$ and $C(l_n)$ are the processing capacity required to execute j_{new} and l_n respectively. u is the processing capacity of the system per unit time.

Thus, the available processing capacity of the system in I_n, $u * \sigma$, should not be exceeded by the processing capacity required to execute $C(j_n)$, $C(l_n)$ and $C(j_{new})$. The processing capacity requirement is measured by the number of CPU cycles. The processing capacity needed to execute a service is the number of busy CPU cycles spent on executing a request of that service.

In practice, achieving a 100% CPU utilisation is not realistic. Abdelzaher and Lu [178] have derived an upper utilisation bound for schedulability of requests in a high-volume real-time service, such as a busy web server. They have found that a set of scheduled tasks will meet their deadline if system utilisation does not exceed 0.58. Thus, $u * \sigma$ in the equation above should be reduced to reflect this practical consideration.

5.6.2 Scheduling Strategy

As described in Section 5.6.1, the jobs already scheduled in a time interval I_n and the forecast of possible new requests in that interval are sufficient parameters to decide whether it is possible to schedule a new job j_{new} with a specific service time in I_n. However, what if j_{new} can also be scheduled in I_{n+1}, should j_{new} be scheduled in I_n or in I_{n+1}? A possible scheduling strategy could take into account customer-oriented parameters in deciding when to schedule service requests. Such parameters could include the history of customer visits pattern, customer time zone and other customer details including age, occupation, etc. However, it was decided in the current system to use a simpler strategy in which earliest time intervals are given scheduling priority. This ensures that a postponed customer request is executed (if the customer returns for discount offer fulfilment) as soon as system resources can cope with such a request.

5.6.3 Forecasting System Load

As described in Section 5.6.1, forecasting of system load at a future interval is taken into account when scheduling postponed jobs in that interval. For forecasting purposes, only system load caused by new customer requests is considered. Load imposed on the system by scheduled customer requests returning for CBAC offer fulfilment is not included. Such scheduled load is accounted for in solving the scheduling problem using $C(j_n)$ as described in Section 5.6.1. In the rest of this subsection, *'system load'* is used to refer only to load caused by new customer requests rather than scheduled requests.
Prediction of future system load is based on system load history. If time is split into discrete equal (e.g. one hour in length) intervals

$$I_{-n}, I_{-n+1}, I_{-n+2}, \ldots, I_0, \ldots, I_{n-2}, I_{n-1}, I_n$$

where time represented by intervals I_{-n} to I_{-1} is the history of system runtime, I_0 is the current time interval and intervals I_1 to I_n are future intervals in which jobs are scheduled.
As requests are executed in I_0 a record of the system cumulative load h_0 is maintained, that is, the amount of processing capacity that has been required to execute new requests arriving during that interval. $h_{-n}, h_{-n+1}, \ldots, h_{-1}$ represent records of cumulative load in the past intervals $I_{-n}, I_{-n+1}, \ldots, I_{-1}$. It is assumed that there is a repeating pattern of workload over a time period. System load in a future interval is predicted from the cumulative load of a corresponding interval in history. The correspondence between future and history intervals is determined by CBAC's forecasting time period. For example, if CBAC's forecasting interval is a week, then the difference between future and history intervals is a multiple of seven days.
This repeating pattern behaviour is consistent with Internet Commerce workload's *self-similarity* property described in Section 2.4.5 where the traffic structure is the same. Regarding load bursts, a pattern can also be found such as in certain hours of the day, e.g. lunchtime. Load size variations can also be noticed across days, e.g. a weekday load is usually different from that at the weekend. The forecasting time period could be specified in CBAC setup to be a day, a week or longer depending on the application and the nature of its workload.

5.6.4 Scheduling Contribution to Cost

Scheduling contributes to determining the amount of discount/charge A as described in Section 5.4. If a service request arrives in time interval I_n, the jobs scheduled for execution in this interval j_n contribute to calculating A. The scheduling factor S is defined as:

$$S = \frac{C(j_n)}{u * \sigma}$$

That is, the fraction of reserved processing capacity for executing jobs already scheduled in I_n. Although $C(l_n)$, the system processing capacity required to execute forecasted (not scheduled) load is not included in the calculation of S, it influences its value indirectly by contributing to the scheduling process as described in Section 5.6.1.

5.7 CBAC-specific Services

For an Internet Commerce system to support CBAC, a set of additional services must be provided by its application server. They are CBAC-specific services which are essential to support CBAC's operations. Three groups of CBAC-specific services are required to:

- Send an offer containing alternative options to a customer.
- Capture the customer decision on the offer.
- Fulfil postponed requests.

The navigational pattern through such services is governed by customer behaviour when interacting with CBAC.
Figure 5.5 illustrates the CBAC-specific services that should be offered by an Internet Commerce application that adopts CBAC. The Admission Control service which is required for preparing and sending a CBAC offer to a customer is described in Section 5.7.1. This is followed in Section 5.7.2 by a description of the services required to capture customer decision which are Accept Discount, Accept Charge and Reject Offer. The Back for Offer and Fulfil Offer services, needed to allow the customer to resume a postponed request, are discussed in Section 5.7.3. Finally in Section 5.7.4, a discussion of customer behaviour when dealing with CBAC is provided including how such behaviour can be modelled.

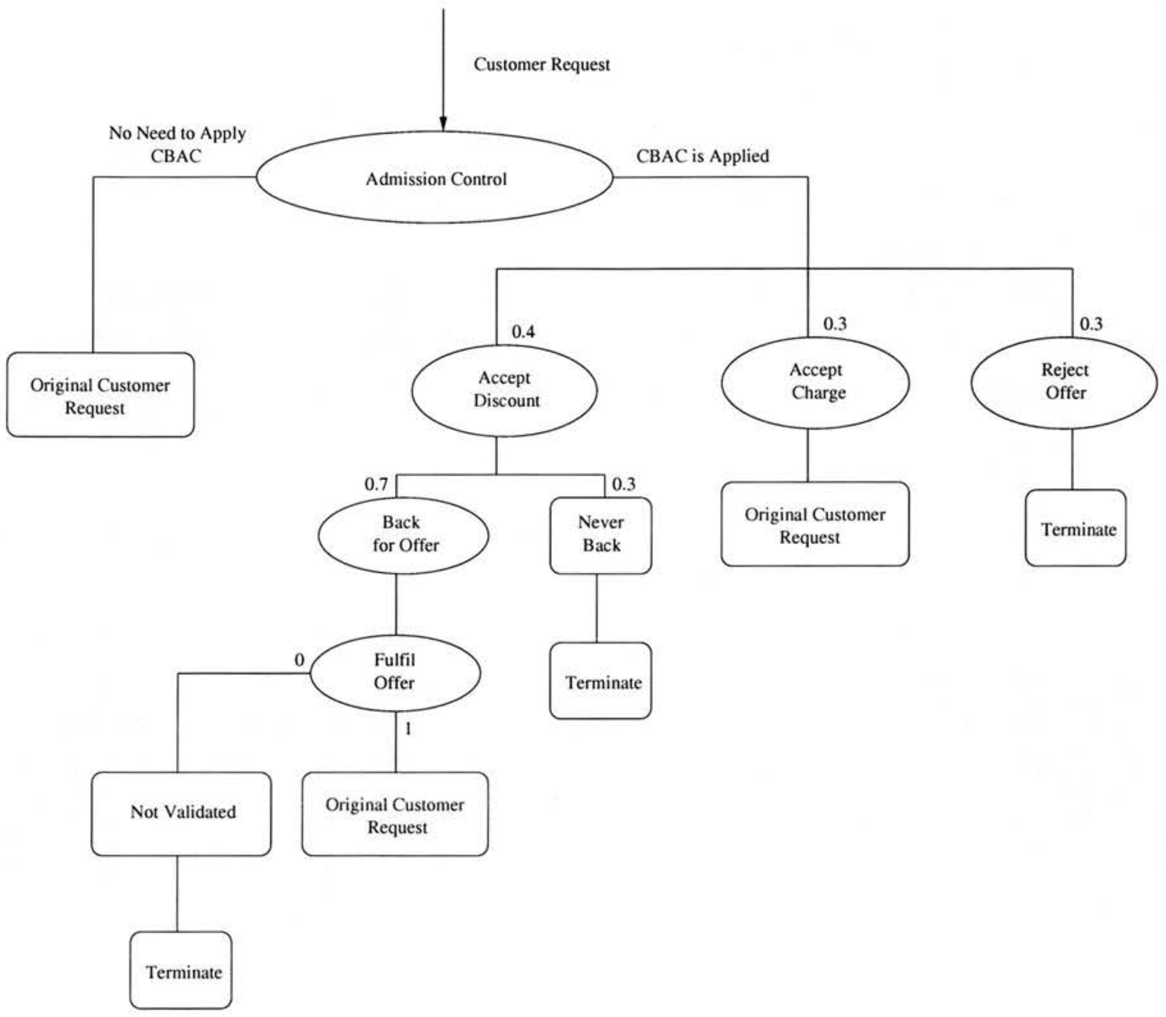


Figure 5.5: CBAC-specific Services

5.7.1 Making a CBAC Offer

The Admission Control front-end service shown in Figure 5.5 is responsible for implementing CBAC's admission policy. By monitoring system load, it decides whether CBAC is applied or not when each customer request is received. If the system's used resources are less than CBAC's threshold ϵ (see Section 5.4.2), the customer request is executed directly without making any CBAC offer. However, if the system's used resources are equal or greater than ϵ, Admission Control uses CBAC's discount-charge model, CBAC's navigational model and scheduling mechanism, described in Section 5.4, Section 5.5.3 and Section 5.6 respectively, to construct a CBAC discount/charge offer. The offer constructed is sent to the customer as a response to their original request.

5.7.2 Capturing Customer Response

A CBAC offer gives the customer three options to choose from. Firstly, the customer can decide to accept the discount specified in the offer and postpone the original request to a later offered scheduled time. The customer should use the Accept Discount service shown in Figure 5.5 to select this option. A response is then sent to the customer which includes an offer Id and a confirmation of the scheduled time at which the customer should come back for offer fulfilment. A possible extension to CBAC is to offer the customer a choice of return time with different discounts based on the load of the future time interval offered or how far it is in the future. For that, a more sophisticated scheduling mechanism should be used which contributes to an extended version of the discount/charge model described in Section 5.4.2.

Alternatively, the customer can choose to accept to pay an extra charge specified in the offer to execute the original request immediately. The customer should request the Accept Charge service shown in Figure 5.5 to select this choice. The customer will then be allowed to proceed in interacting with the system starting from the original request.

The last option given to the customer is to reject the offer completely. The customer selects this option by requesting the Reject Offer service shown in Figure 5.5. Making this choice would cause the customer session to be terminated.

5.7.3 Fulfilling Postponed Offers

A customer who has chosen to accept the discount option of a CBAC offer can resume interaction with the system by requesting the Back for Offer service shown in Figure 5.5. It was decided to delegate the task of initiating the offer fulfilment process to the customer, so that no extra load is introduced on the server reminding customers of their return times to the site. Using the Back for Offer service, the customer should enter the offer Id provided when the decision was made to accept the discount. The Fulfil Offer service is then used to validate the offer details for the given offer Id. If the offer details are validated, including the return time for the customer, Fulfil Offer directs the customer to request the original request and the customer navigation is resumed. However, if offer Id is not validated or the offer scheduled time has passed, the customer is permitted to request the Home service starting a new session and the offer claim is rejected. When requesting fulfilment of an offer at a time that does not match the offer scheduled time, the customer is not allowed to resume system navigation and is informed to return at the right time if it is still to come.

5.7.4 Customer Behaviour

There are many psychological and social factors that could affect customers' behaviour when interacting with a CBAC-enabled system. Customer personal details such as age, gender, wealth, computer literacy, occupation and cultural background are factors likely to influence customer reaction to CBAC. Investigating such factors and how they determine customer access patterns when dealing with CBAC is beyond the scope of this thesis and should be considered in further research as outlined in Chapter 7.

The service offered by a CBAC-enabled system should have an effect on the customer's behaviour when dealing with CBAC interactions. Different Internet Commerce application types will result in different CBAC customer behaviour. Thus for each type of application, the history of system navigation needs to be analysed to tune CBAC parameters in order to improve customer reaction to CBAC and thus achieve optimal performance. Due to the wide variety of Internet Commerce applications and thus the varying possible user reaction to CBAC, scenarios of possible user reactions were assumed throughout this thesis to analyse CBAC performance effects. One CBAC access pattern scenario was assumed to follow the navigation probabilities shown in Figure 5.5. It was assumed that 40% of customers accept a discount of which only 70% come back at the future offer scheduled time for offer fulfilment. It was assumed that no customers return for offer fulfilment at the wrong time or fail offer validation. 30% of customers were assumed to select the accept charge option and 30% completely reject the offer. As explained in Chapter 6, experiments were performed to investigate how this navigation pattern and another variation of it affected performance metrics.

5.8 CBAC Applications

To determine the applicability of CBAC and its optimal parameters for an Internet Commerce system, a group of factors which are dependent on the application type should be considered. Firstly, whether the application is of time-critical nature or not. For example, consider a train booking Internet Commerce system. In such a system, the duration between train ticket purchase time and train departure time should be taken into account when CBAC is used. Given such timing constraints, CBAC's application-oriented scheduling could be used to enhance customer satisfaction. Business decisions need to be made on the extent to which the urgency of user requests should be exploited to increase profit. Those decisions and others will be reflected in choosing CBAC's parameter values.

Another factor that needs to be considered when using CBAC is the application's user population. User behaviour when dealing with CBAC offerings should be taken into account when setting up CBAC. Customer background, age, sex, etc. could aid in predicting navigational patterns through CBAC-specific services. Customer behaviour can also be "learned" dynamically and CBAC parameters could be adapted accordingly.

The type of application could also influence business managers considering adopting CBAC. For CBAC to be used, changes in the application's business model are required. The extent of those changes could vary depending on the characteristics of the application under consideration.

5.9 Summary

In this chapter, CBAC, a dynamic software technique to enhance Internet Commerce QoS, is described. Without relying on customer request rejection, it manages system congestion by using a dynamic pricing method, that is sensitive to current system load and navigational structure, to encourage customers to postpone their requests. CBAC provides a comprehensive framework for Internet Commerce QoS enhancement in terms of the performance metrics it optimises. It is designed to provide service times competitive with low load situations by ensuring that system load is distributed to less loaded future time intervals. The charging element of CBAC and abandoning request rejection, to increase customer satisfaction, is designed to maximise profit. Scheduling customer requests in lightly loaded time periods increases system utilisation, sustains throughput and reduces request failure.

In the next chapter, an investigation of CBAC performance is described. ECBench is extended to support CBAC benchmarking. Experiments have been designed and performed to examine the effect of CBAC on different Internet Commerce QoS metrics.

Chapter 6

Investigating CBAC Performance

6.1 Introduction

This chapter provides an analysis of the QoS enhancement achieved by using CBAC. The impact of CBAC on service time, system utilisation, throughput and profit is investigated. The analysis was carried out using the ECBench tool described in Chapter 3 and provides a demonstration of how the tool could aid a researcher in understanding the performance impact of a novel technique such as CBAC.

The rest of this chapter is structured as follows. The extension made to ECBench to support CBAC benchmarking is discussed in Section 6.2. This is followed in Section 6.3 by a description of the analysis done on CBAC's performance, comparing it with a CBAC-disabled system and investigating the effect of different CBAC parameters on performance. Finally, in Section 6.4, a summary is given.

6.2 ECBench Support for CBAC

To enable the analysis of CBAC's performance effect on an Internet Commerce system, the ECBench tool has been extended to support the benchmarking of CBAC. These extensions are described in the following subsections.

6.2.1 ECBench Server-side Extensions

ECBench's Application Server, implemented using Java Servlets and described in Section 4.4.2, was extended to support CBAC. Figure 6.1 illustrates the main servlet and non-servlet classes composing the CBAC extension. Central to the extension is the CBACServlet component which is composed of two main parts, the Service Map and Scheduler. Service Map implements the navigational model of CBAC, CBAC-NM, described in Section 5.5.3, including its service time estima-

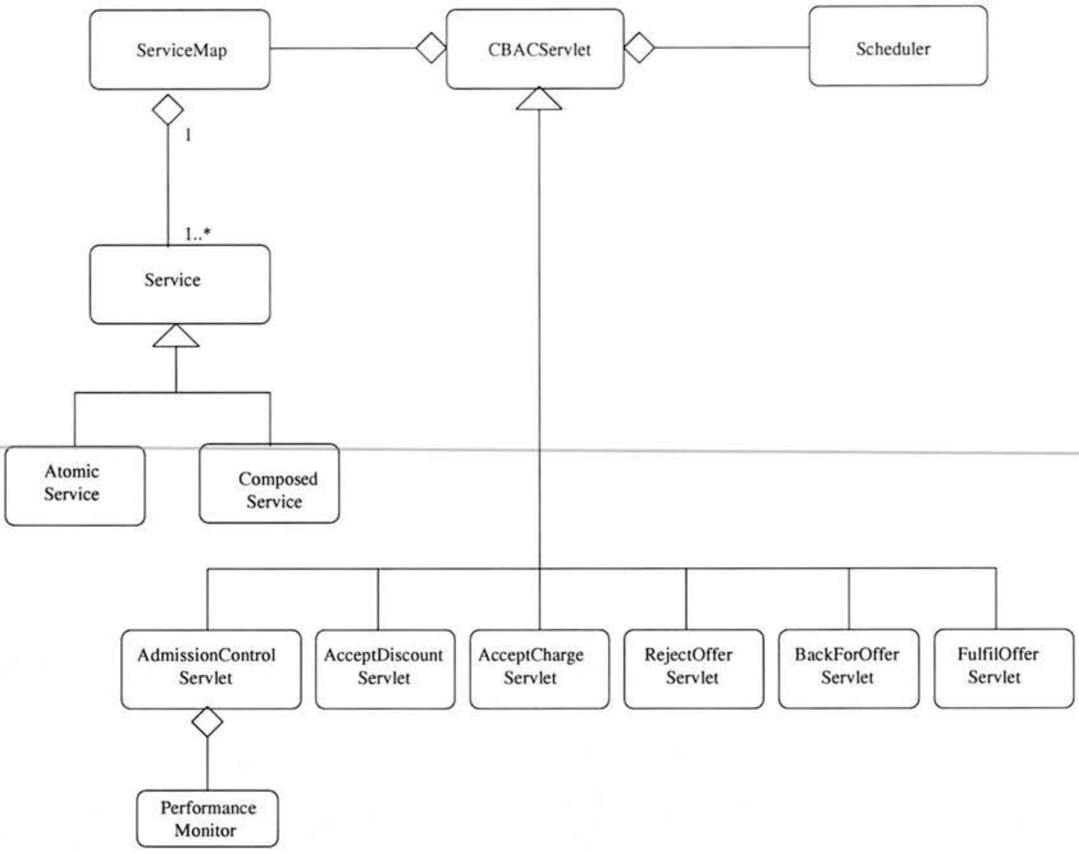


Figure 6.1: ECBench's Application Server Extension

tion described in Section 5.5.4. The AtomicService and ComposedService classes inherit the Service class. A list of Service objects is held by the Service Map representing different services in CBAC-NM.

The Scheduler class is responsible for implementing CBAC's scheduling mechanism described in Section 5.6. It interacts with a simple CBAC database which is hosted by ECBench's Database Server. An entity-relation diagram for the CBAC database is shown in Figure 6.2. The Scheduler selects the time interval to be offered to a customer accepting a discount (during which the offer can be fulfilled). If the customer then decides to accept the discount offer, the Scheduler updates the database, adding a job to the JobsScheduled table. A job is identified uniquely by a jobID which corresponds to the offer Id that is required for offer fulfilment. Each job in the JobScheduled table is associated with a SchedulingInterval table record by setting time to reference the appropriate intervalID. The JobsScheduled table contains other job information including the first service which the customer will request when returning for offer fulfilment, the discount offered to the customer, the estimated number of CPU cycles required in the customer session and

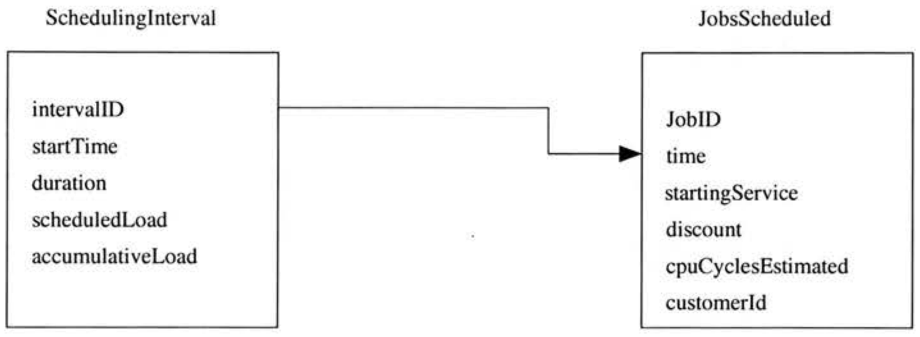


Figure 6.2: CBAC Database Entity-relationship Diagram

the customer identification. In order to minimise database interactions overhead, the Scheduler maintains information about the scheduled load and historical load for each time interval. These are required for deciding when to schedule a job. Due to memory size limitations (especially if CBAC is used over a long time period), a persistent copy of such load information is maintained by periodically updating the SchedulingInterval table.

A Java Servlet is used to implement each of the CBAC-specific web interactions described in Section 5.7. All the servlets inherit CBACServlet and thus share its ServiceMap and Scheduler. A description of each servlet is given below:

- AdmissionControlServlet: As described in Section 5.7.1, all customer requests go through this servlet to check whether CBAC is to be applied. The AdmissionControlServlet implements CBAC's pricing model which was explained in Section 5.4. Its Performance Monitor object periodically checks CPU usage to determine when the threshold ϵ is exceeded in order to activate CBAC. The formation of CBAC offers is performed by the AdmissionControlServlet querying the ServiceMap and the Scheduler for the information required to calculate the discount/charge amount A and the proposed return time for the discount part of the offer. Java Servlets' session tracking mechanism (described in Section 4.3) is used to temporarily store the pending offer.

- AcceptDiscountServlet: This servlet is requested when a customer accepts the discount part of an offer. The servlet examines the customer session to check the offer details. The Scheduler is then informed to add a job, at the offer scheduled interval, to the JobScheduled table of the CBAC database described above. The JobID is sent to the customer as the offer Id to use when coming back for offer fulfilment.

- AcceptChargeServlet: As in AcceptDiscountServlet, the offer details are retrieved from the customer session. The amount of extra charge that the customer has to pay is kept in the customer session to be added later to the customer's final bill. The customer is then allowed to proceed with their original request.

- RejectOfferServlet: This servlet terminates the customer session and deletes the offer pending details.

- BackForOfferServlet: This servlet contains a form to allow a returning customer, who was given and accepted a discount offer, to enter the offer Id.

- FulfilOfferServlet: This servlet requires offer Id in order to check the offer validity by querying the CBAC database. If the offer is valid, the discount amount offered is stored in the customer's session to be deducted from the final bill. The customer is then directed to proceed with their original request. If the offer is not validated, the customer will be permitted to start a new session without any discount.

6.2.2 ECBench Workload Extension

The workload part of ECBench, described in Section 3.6, was extended to support the handling of CBAC interactions. The following was added to the EB component of ECBench's RBE:

- Several classes to represent CBAC-specific interactions including, Accept Discount, Accept Charge, Reject Offer, Back For Offer and Fulfil Offer were implemented. The classes inherit Web Interaction, described in Section 3.6.1, which enables the EB to navigate through CBAC services according to a specified access pattern determined by navigation thresholds.

- Another two classes inheriting Web Interaction were also added to EB: Offer Decision and Terminate. They are dummy interactions that are never requested. OfferDecision represents the EB in the stage of making a decision about whether to take the discount, accept the charge or reject the offer according to specified navigation thresholds. Terminate represents the EB at the end of the navigation session.

- Offer Details, representing details of a CBAC offer accepted including time of return and offer Id. The EB uses the timing information to wait for the required time before making a BackForOfferServlet request. The offer Id is used in the offer fulfilment stage when the EB requests FulfilOfferServlet.

6.3 CBAC Performance Analysis

This section provides a description of the work done to investigate the feasibility of CBAC and its effectiveness in enhancing QoS in an Internet Commerce system. ECBench was used to carry out this investigation based on its extension described in Section 6.2 and its Java Servlets application server implementation described in Section 4.4.2. Table 6.1 is a shortlist of CBAC parameters derived from CBAC's description provided in Chapter 5. These parameters are used throughout this chapter in the analysis of CBAC performance.

The rest of the section is structured as follows. The design and deployment of the experiments performed are described in Section 6.3.1. Analysis of the results obtained with respect to service time, system utilisation, throughput and profit metrics is provided in Sections 6.3.2, 6.3.3, 6.3.4 and 6.3.5 respectively. CBAC's overhead is investigated in Section 6.3.6, and the effect of CBAC's load forecasting is described in Section 6.3.7.

Parameter	*Description*
ϵ	CBAC threshold.
β	Delay weight.
δ	Scheduling weight.
α	Load weight.
A_{max}	Maximum limit on discount/charge offered.
C	Charging factor controlling the ratio between charge and discount.
Navigational structure	Details of services offered by the system including how they are connected, navigational probabilities between them and estimations of the required service time for each of them.
L	The number of items the customer is likely to request.
CBAC access pattern	Reflecting user behaviour when dealing with CBAC. Probability of navigation through the following services is required: Accept Discount, Accept Charge, Reject Offer, Back For Offer and Fulfil Offer.
Workload	The size and structure of workload.
Scheduling interval	The length of the unit interval in which CBAC's postponed requests are scheduled.
Forecasting period	The length of the forecasting period over which history data is re-used.

Table 6.1: Shortlist of CBAC Parameters

6.3.1 Experiment Design and Deployment

Six experiments were designed and performed, the first is *CBAC-disabled* in which CBAC was not applied. In the other five experiments, *CBAC1*, *CBAC2*, *CBAC3*, *CBAC4* and *CBAC5*, CBAC was applied with different parameter sets as detailed in Table 6.3.

The duration of each of the experiments was 48 hours. Two day experiments were performed to investigate the effect of one day load forecasting on CBAC performance. In all experiments, ECBench's workload was toggled every hour between two levels, high and low, as illustrated in Figure 6.3. At high load, 248 EBs were used. Two EBs were used to represent the low load. In a real system, a pattern can usually be found of high and low workload at certain times, e.g. high load at lunchtime, low load after midnight, etc. The workload used in the experiments represented an extreme case of such a repetitive pattern in which the pattern is repeated every two hours. Such simplified workload was chosen to analyse system performance when stressed at high intervals and during low load intervals.

The workload used in the experiments represents a fixed user population. In a high load interval, a total fixed population of 248 users were interested in the service offered by the system at anytime during the experiment. Similarly in a low load interval, the population of users was two. Such users were represented by EBs continuously generating service requests, thus producing a large number of jobs during the course of the experiment. Initially, at the start of each hour, the number of active EBs is either 248 or 2 depending on the level of workload in the hour. In the *CBAC-disabled* experiment, this user population was maintained throughout the remainder of the hour. However, in CBAC experiments, when a CBAC discount offer was accepted (in a high load interval), the EB through which this acceptance occurred becomes inactive until the end of the interval. Thus, the number of EBs through which a discount offer acceptance occurs are removed temporarily from the original workload. It was decided to deactivate such EBs in order to test the effect of CBAC's discount core feature and the reduction of system workload it causes.

The navigational structure described in Section 5.5.2 and depicted in Figure 5.2 was used in all the experiments. A preliminary experiment was performed in order to form service load estimates for each of the different interactions in the

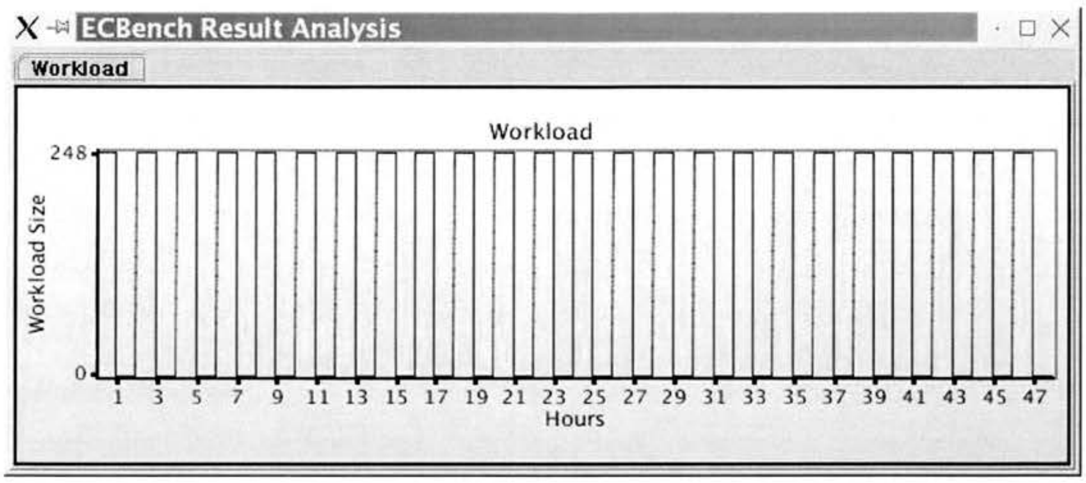


Figure 6.3: Common Workload to All Experiments Performed

Service	*Total Time Estimate (s)*	*Idle Time (s)*
Home	0.147	0
Best Sellers	0.5	0.43
New Products	0.235	0.009
Search Request	0.12	0
Search Result	0.128	0.012
Product Details	0.125	0.001
Shopping Cart	0.131	0.012
Customer Registration	0.127	0
Buy Request	0.707	0.589
Buy Confirm	0.171	0.048

Table 6.2: Service Load Estimates

structure. By using a workload size of one EB, delays incurred by request queueing were eliminated and thus, the time estimates acquired reflected the actual service load of each interaction. For this experiment, the Java Servlets implementation for the navigational structure interactions (described in Section 4.4.2) was used and deployed in an Apache Tomcat web server. Both ECBench's Web Server and Database Server were deployed on the same machine. ECBench's RBE, which consisted of only one EB, was deployed on another machine. The time estimates acquired for each web interaction are listed in Table 6.2. The *Time Estimate* measurement was considered as the time elapsed between the server receiving a web interaction request until a response is returned. Part of the *Time Estimate* measurement is the *Idle Time* during which request processing is suspended waiting for database operation to be completed. The *Idle Time* component was measured for each of the web interactions. These time estimates were used in the main experiments for three purposes:

- As delays to model the interactions and abstract their database transactions, HTML page construction and sending. The server multi-threading behaviour was achieved by modelling the interactions' *Idle Time.* This was implemented by suspending service requests (putting request threads into sleep) for a duration equals to the service's *Idle Time* resulting in CPU processing to alternate between different requests.
- Secondly, the estimates were used in the CBAC experiments (mapped to CPU cycles) in order to get total time estimates, required for request scheduling, as explained in Section 5.5.4.
- Finally, the estimates formed a set of service time QoS targets and were used to assess system performance in the main experiments.

In CBAC experiments, the effect of varying the threshold parameter ϵ, the charging factor C and the CBAC-specific service access pattern was investigated. The values of these parameters are listed in Table 6.3 for each experiment. In *CBAC1, CBAC2, CBAC3,* ϵ was varied to examine the effect of CBAC's strictness in controlling system load. CPU utilisation was used as an indicator of system load.

Experiment	ϵ	C	*Accept Discount*	*Accept Charge*	*Reject*	*Back For Offer*	*Not Back*
CBAC1	0.58	3	0.4	0.3	0.3	0.7	0.3
CBAC2	0.7	3	0.4	0.3	0.3	0.7	0.3
CBAC3	0.78	3	0.4	0.3	0.3	0.7	0.3
CBAC4	0.78	3	0.4	0.5	0.1	0.9	0.1
CBAC5	0.78	6	0.4	0.5	0.1	0.9	0.1

Table 6.3: CBAC Experiments Varied Parameters

Parameter	*Value*
β	0.33
δ	0.33
α	0.33
A_{max}	1
L	1
Scheduling interval	1 hour
Forecasting period	1 day

Table 6.4: CBAC Experiments Common Parameters

A CPU reading was taken every 20 seconds to determine whether ϵ had been exceeded and whether to apply CBAC. Taking a CPU reading for every request would have increased CBAC's overhead. In *CBAC4*, CBAC's customer access pattern was varied increasing the percentage of customers accepting a charge, reducing the offer rejection rate and increasing the percentage of customers coming back for a discount offer. In all CBAC experiments, the worst case scenario was adopted in which all customers who rejected an offer immediately attempt to request a new service. The effect of increasing the charging factor C on profit was investigated in *CBAC5*. The rest of the CBAC parameters were common to all CBAC experiments and are listed in Table 6.4.

In all experiments, ECBench's workload at high intervals was distributed equally across four machines. Each of the machines was running Linux and had an Intel Pentium 4, 1.80GHz processor. The server was deployed on an Apache Tomcat server running on an Intel Pentium III 1GHz Linux machine. The same methods described in Section 4.5.3 were used to collect system measurements. As a scheduling interval of one hour is used in all CBAC experiments (see Table 6.4), the amount of system resources available per scheduling interval on the server machine was $3.6 * 10^{12}$ cycles. Such a resource estimate was required to decide when to schedule postponed customer requests as described in Section 5.6.1. It was assumed that only $3.6 * 10^{9}$ cycles were the available resources each hour. One reason for restricting the available resources for scheduling was because a proportion of CPU cycles were consumed in system-related processes. However, the main reason for this restriction was due to the limited size of workload in the experiments performed and the large amount of available resources per scheduling interval. Under estimating the resources available at each interval enables the distribution of scheduled jobs across future intervals and thus provides a better distribution of the load.
Analysis was performed to investigate how far in the future a user request is scheduled. Table 6.5 and Figure 6.5 show the number of jobs scheduled at each scheduling interval in a CBAC experiment performed with the same parameters as *CBAC5* (see Table 6.3 and Table 6.4). The duration of the experiment was 30 hours. The workload of the experiment is shown in Figure 6.4. It is subject to the same workload structure as the main experiments described in this chapter. The scheduling strategy described in Section 5.6.2 was used in which the earliest time intervals were given scheduling priority. This scheduling strategy was also used in all other experiments described in this chapter. The variation in the number

Time Interval	Workload Size	No. of Jobs Scheduled into this Interval
1	248	0
2	2	235
3	248	22
4	2	216
5	248	58
6	2	203
7	248	71
8	2	211
9	248	35
10	2	213
11	248	50
12	2	213
13	248	50
14	2	208
15	248	69
16	2	215
17	248	64
18	2	208
19	248	46
20	2	215
21	248	57
22	2	226
23	248	27
24	2	208
25	248	46
26	2	212
27	248	57
28	2	212
29	248	49
30	2	211

Table 6.5: Number of Jobs Scheduled in Each Hour

of jobs scheduled in each interval is due to the varying size of the jobs scheduled. The size of each job is determined by the service time estimate of the request at which a CBAC discount offer is accepted and all subsequent requests likely to be requested (as described in Section 5.5.4). It can also be seen from Figure 6.5 that the number of jobs scheduled at each of the low load intervals is not substantially less than the load imposed on the system from the previous interval (a high load interval). This indicates that a large proportion of jobs were scheduled in the next interval. The cumulative scheduled jobs is depicted in Figure 6.6. It shows that scheduling reaches a regularity quickly, three hours after the start of the experiment.

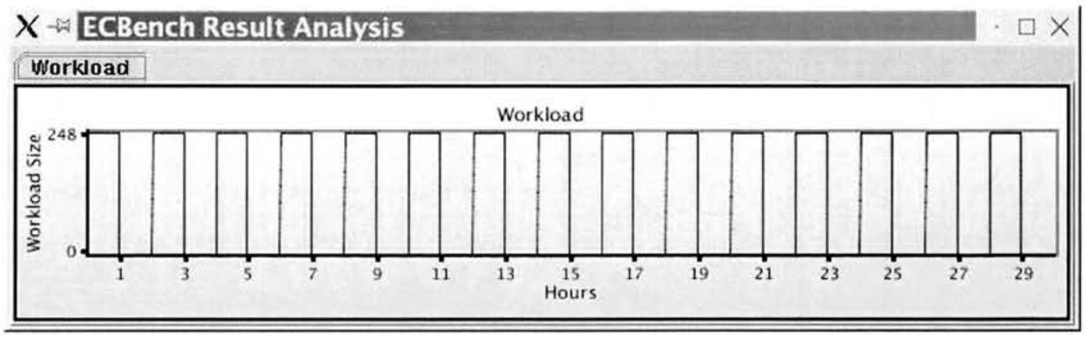


Figure 6.4: Scheduling Effect Experiment Workload

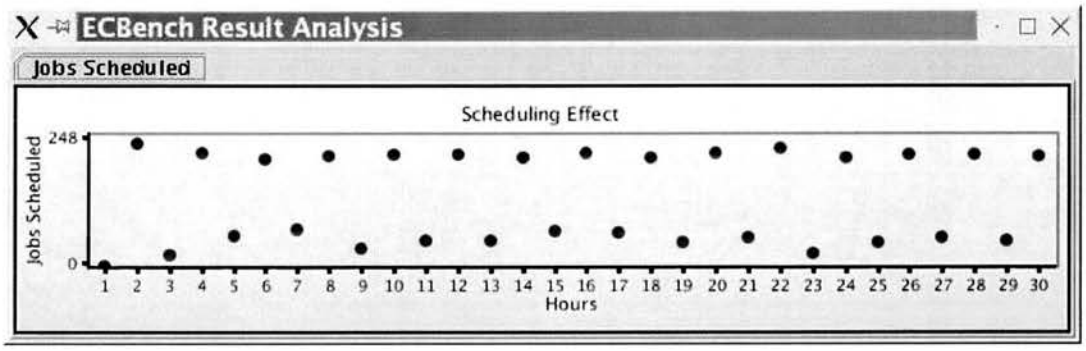


Figure 6.5: Jobs Scheduled at Each Interval

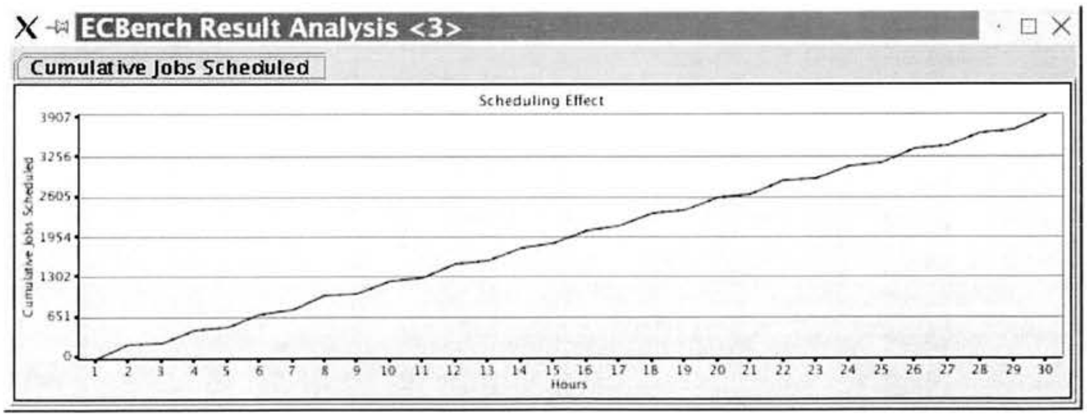


Figure 6.6: Cumulative Jobs Scheduled

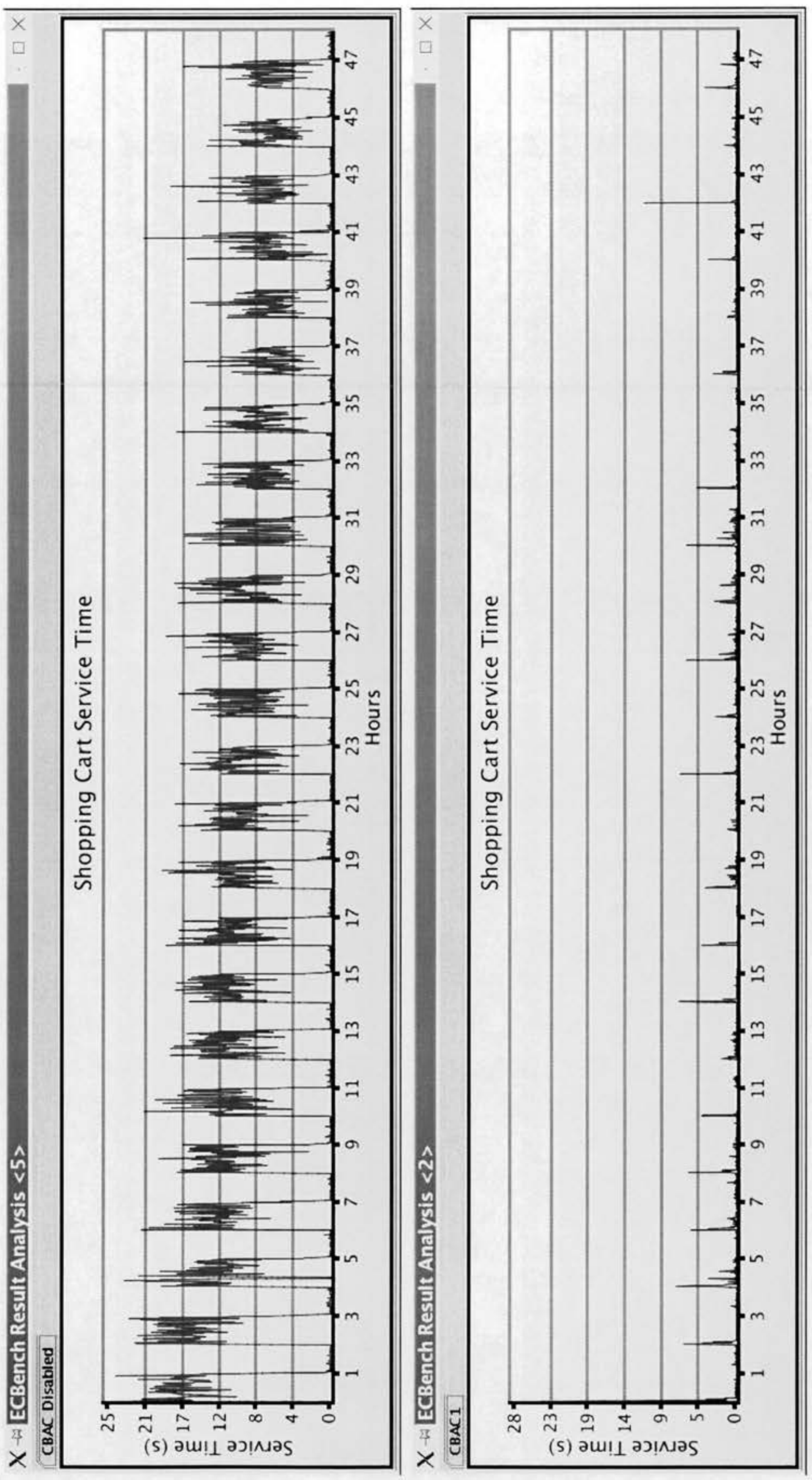
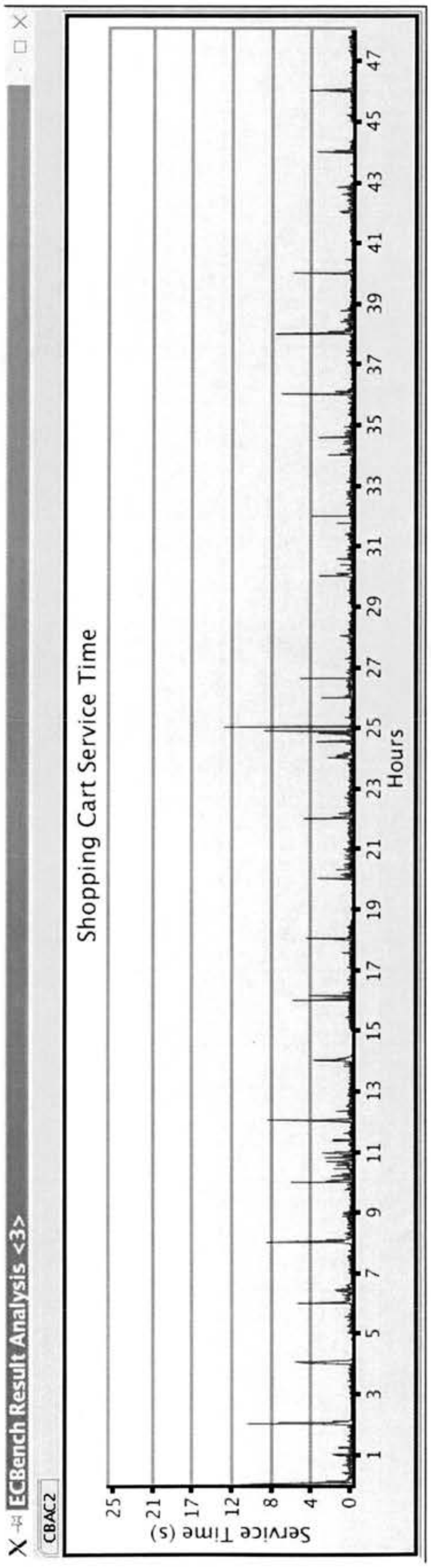


Figure 6.7: *CBAC-disabled, CBAC1, CBAC2* and *CBAC3* Service Times for Shopping Cart



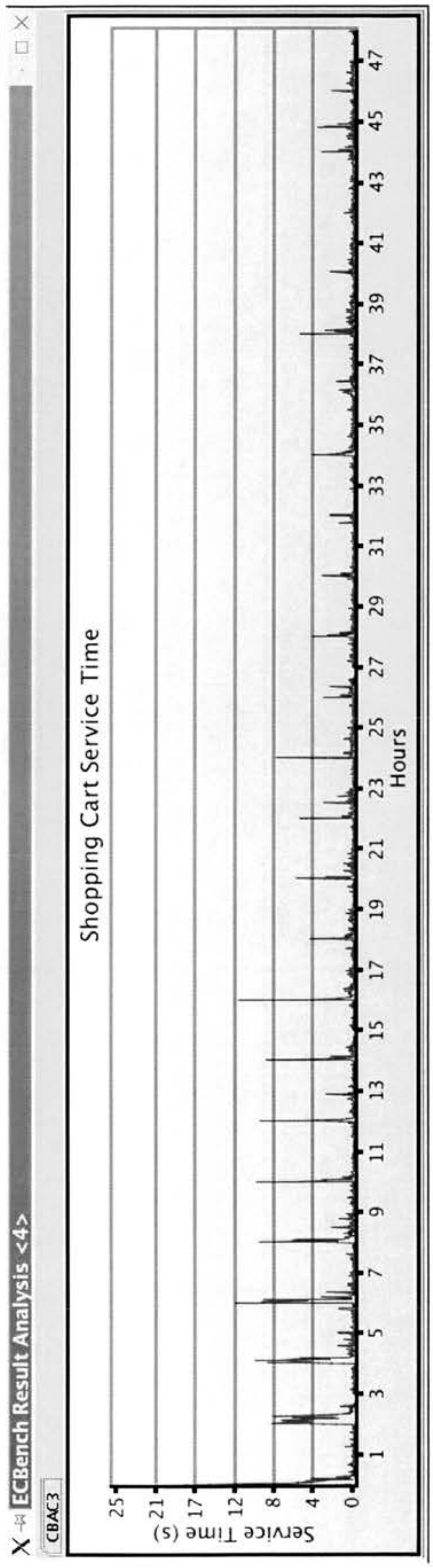


Figure 6.7: *(Cont.) CBAC-disabled, CBAC1, CBAC2* and *CBAC3* Service Times for Shopping Cart

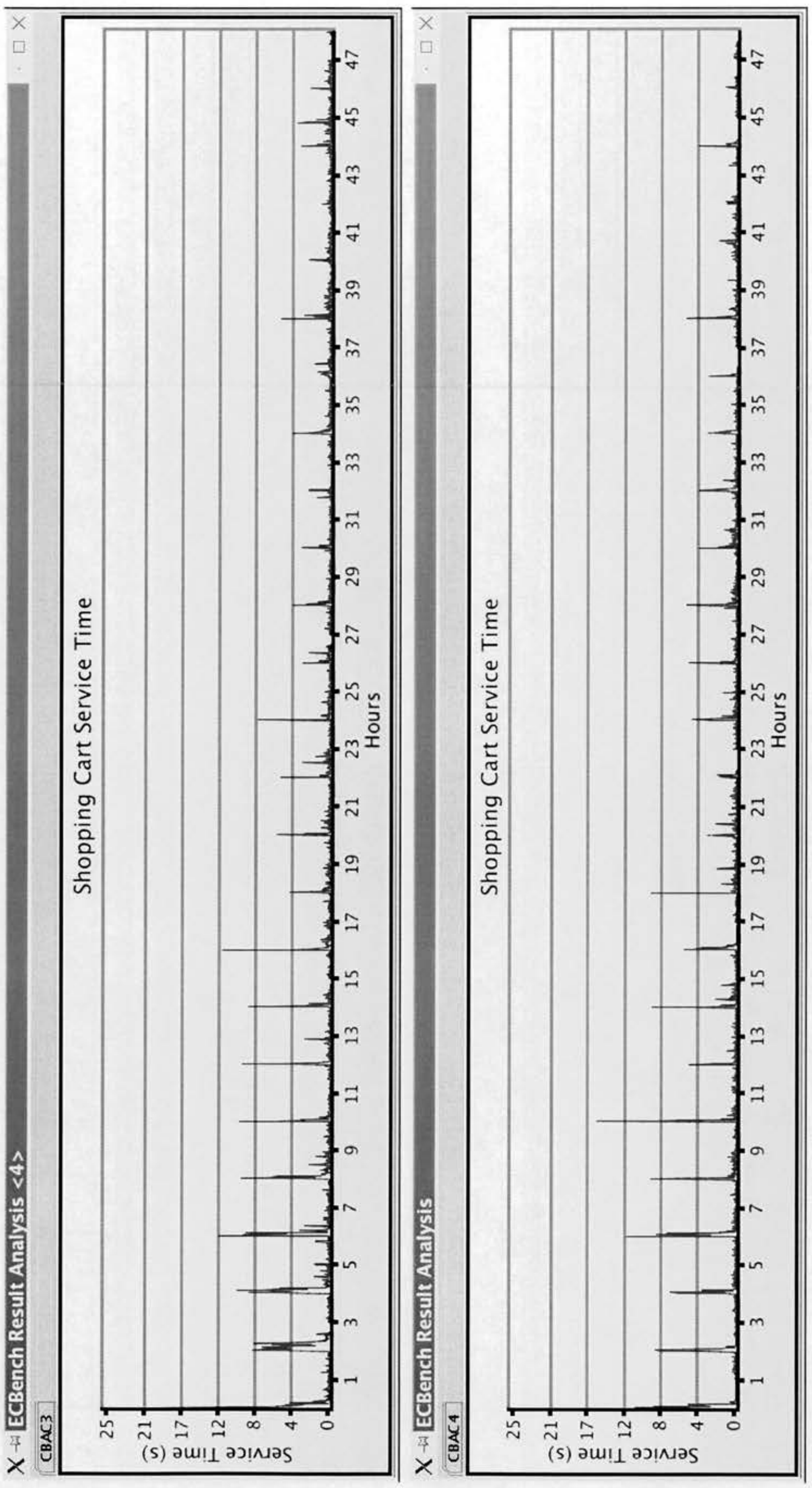


Figure 6.8: *CBAC3* and *CBAC4* Service Times for Shopping Cart

6.3.2 Service Time

As explained in Section 6.3.1, service time estimates were used to model the delay incurred on the ECBench server to fulfil each web interaction in the navigational structure. As these delays were obtained under low workload, they represent a desired QoS level in terms of the service time metric. The extent to which these service times were achieved in the experiments performed is discussed in this section. The Shopping Cart interaction service time is used as a sample interaction to illustrate this metric. Other interactions' mean service time and standard deviation analysis is provided in Appendix E
Figure 6.7 depicts the results obtained from examining Shopping Cart's service time in *CBAC-disabled*, *CBAC1*, *CBAC2* and *CBAC3*. It can be seen that there is a substantial service time reduction in CBAC experiments compared with when CBAC is disabled. CBAC service time tends to be high at the start of high workload periods before CBAC is applied to reduce it. CBAC service time improvement is reflected in the service time mean and standard deviation calculated and shown in Table 6.6. The mean service time achieved in the *CBAC-disabled* experiment was 5.4 seconds for Shopping Cart which was reduced about 18 fold in *CBAC1* and, in the worst case, about 11 fold in *CBAC3*. CBAC service time results are much closer to the 0.131 seconds QoS service time target for Shopping Cart explained in Section 6.3.1. Standard deviation values reflect the higher diversity of service times in the *CBAC-disabled* experiment when compared to the CBAC experiments.

The other observation that can be made from Figure 6.7 and Table 6.6 is the variation in service times for *CBAC1*, *CBAC2* and *CBAC3*. As described in Section 6.3.1, these experiments differ in the ϵ threshold value which was varied between 0.58, 0.7 and 0.78 in *CBAC1*, *CBAC2* and *CBAC3* respectively. It can be seen that increasing ϵ causes service times to slightly increase. This is because, at higher threshold values, CBAC is less strict in controlling the system's congestion state which is translated into slightly poorer service times.

Experiment	*CBAC-Disabled*	*CBAC1*	*CBAC2*	*CBAC3*	*CBAC4*
Mean (s)	5.4	0.3	0.37	0.47	0.41
Standard Deviation	5.88	0.89	0.98	1.2	1.02

Table 6.6: Mean and Standard Deviation Service Time Analysis for Shopping Cart

The effect of varying the CBAC services access pattern on service time is analysed comparing the service time results obtained from *CBAC3* and *CBAC4*. As described in Section 6.3.1, the percentage of customers accepting a charge was increased and the likelihood of CBAC offer rejection was reduced in *CBAC4*. This caused a slight reduction in service time as illustrated in Figure 6.8 and Table 6.6. The reason for this reduction is two fold. Firstly, higher rates of accepting the charge mean that CBAC is applied more regularly as less requests are postponed at each application (up to a maximum of every 20 seconds when a new CPU utilisation sample is taken). This causes service time to be reduced earlier in a high workload interval. The second factor is related to the variation in the CBAC offer rejection rate. As explained in Section 6.3.1, the worst case scenario is assumed in which customers rejecting a CBAC offer return immediately to the site to start a new session. The lower rejection rate in *CBAC4* reduces the load introduced by such customer returns which is reflected in a better service time performance.

6.3.3 CPU Utilisation

In all experiments, the average CPU utilisation of ECBench's server is sampled for measurement every minute (this is different from CBAC's CPU sampling every 20 seconds required for its operation). The CPU utilisation results obtained are depicted in Figure 6.9 for *CBAC-disabled, CBAC1, CBAC2* and *CBAC3* experiments. In *CBAC-disabled*, it can be seen that CPU utilisation is just below one in more than one instant and a high utilisation is maintained for the duration of high load intervals. However, in the CBAC experiments, CPU utilisation is reduced to below CBAC's ϵ threshold, apart from at the start of high workload intervals during the time in which CBAC is being applied. A slower reduction of CPU utilisation can be noticed as ϵ is increased in *CBAC2* and *CBAC3*.
It can also be observed that CPU utilisation drops to a very low level at some time intervals in CBAC experiments during which only system background processes were running. The regularity of these intervals decreases as ϵ is increased (only occurring two times in *CBAC3*). During these periods, CBAC has scheduled all workload to future intervals triggered by a previous sample in which CPU utilisation exceeded ϵ. This explains the reduction of such no-activity intervals as ϵ is increased. This observation illustrates the way CBAC exposes unused capacity which can be exploited as the workload size is increased.

6.3.4 Throughput and Failed Requests

The throughput metric is considered to be the rate at which web interactions are completed successfully in one minute. For the CBAC experiments, CBAC-specific interactions are not included in the metric. The throughput results obtained for *CBAC-disabled, CBAC1, CBAC2* and *CBAC3* are shown in Figure 6.10 and the total number of interactions completed during each experiment is shown in Table 6.7. It can be seen that there is a decrease in throughput when CBAC is applied. This is due to the fact that a percentage of requests handled by the system are for CBAC-specific interactions. Another reason for the drop in throughput is the occurrence of the no-activity periods described in Section 6.3.3. This explains the increase in throughput in *CBAC2* and *CBAC3* when the CBAC threshold ϵ is increased. Requests scheduled for execution beyond the duration of CBAC experiments also contribute to the overall reduction in throughput.

Request Failure is another metric considered in the analysis. The total number of failed requests in *CBAC-disabled, CBAC1, CBAC2, CBAC3, CBAC4* and *CBAC5* is shown in Table 6.8. These request failures were caused by the server's automatic rejection of requests when reaching an overloaded state. It can be noticed that request failure drops dramatically when CBAC is used.

Experiment	*CBAC-Disabled*	*CBAC1*	*CBAC2*	*CBAC3*	*CBAC4*	*CBAC5*
Requests Completed	1,044,902	460,482	522,294	744,540	704,984	732,718

Table 6.7: Total Number of Interactions Completed During Experiments

Experiment	*CBAC-Disabled*	*CBAC1*	*CBAC2*	*CBAC3*	*CBAC4*	*CBAC5*
Failed Requests	80649	5813	5701	5626	5712	5778

Table 6.8: Total Number of Failed Requests During Experiments

6.3.5 Profit

A profit per minute metric was used to measure the system's profit, that is the total amount of profit acquired every minute. It was assumed that the basic profit gained on the completion of each customer session (i.e. executing the Buy Confirm

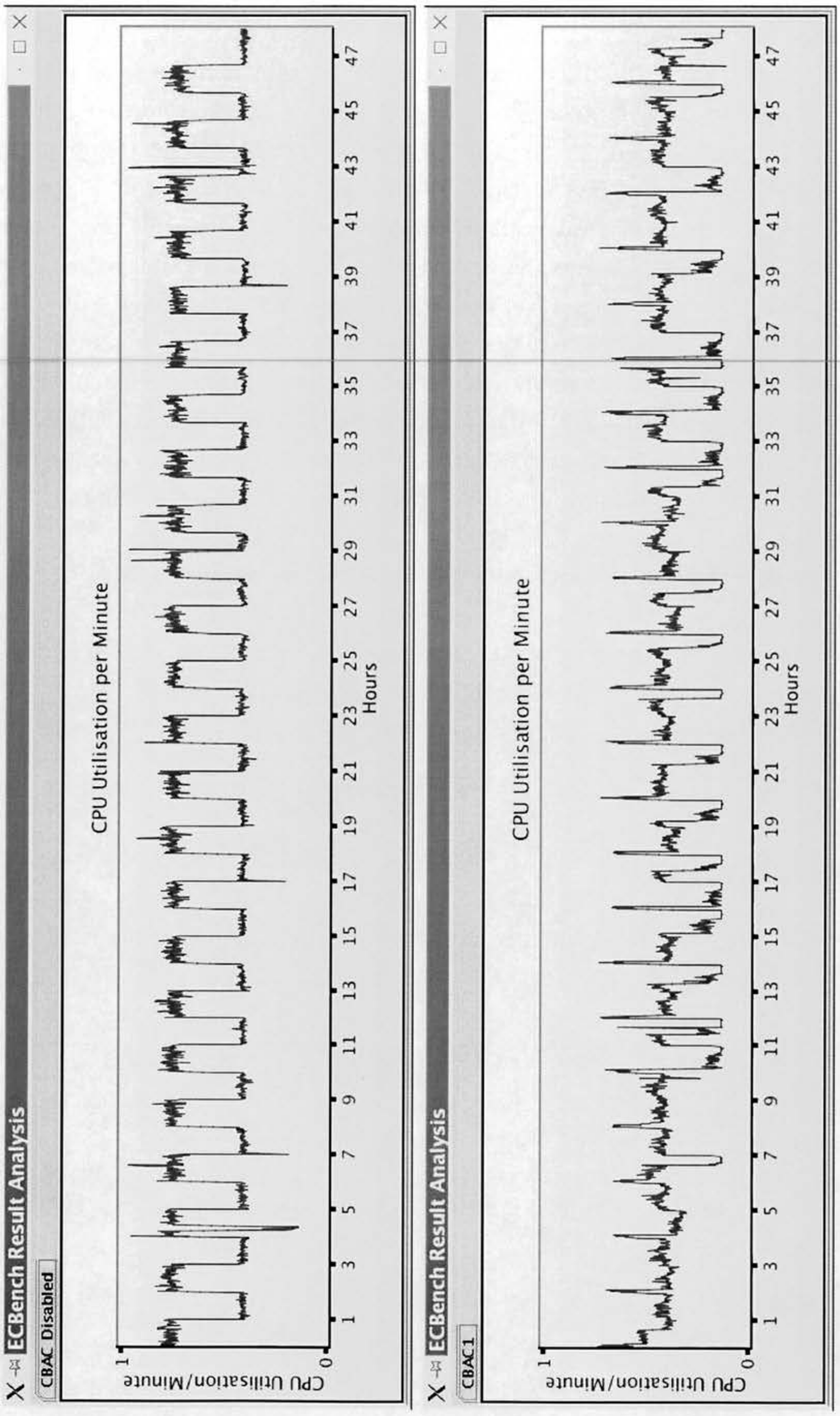
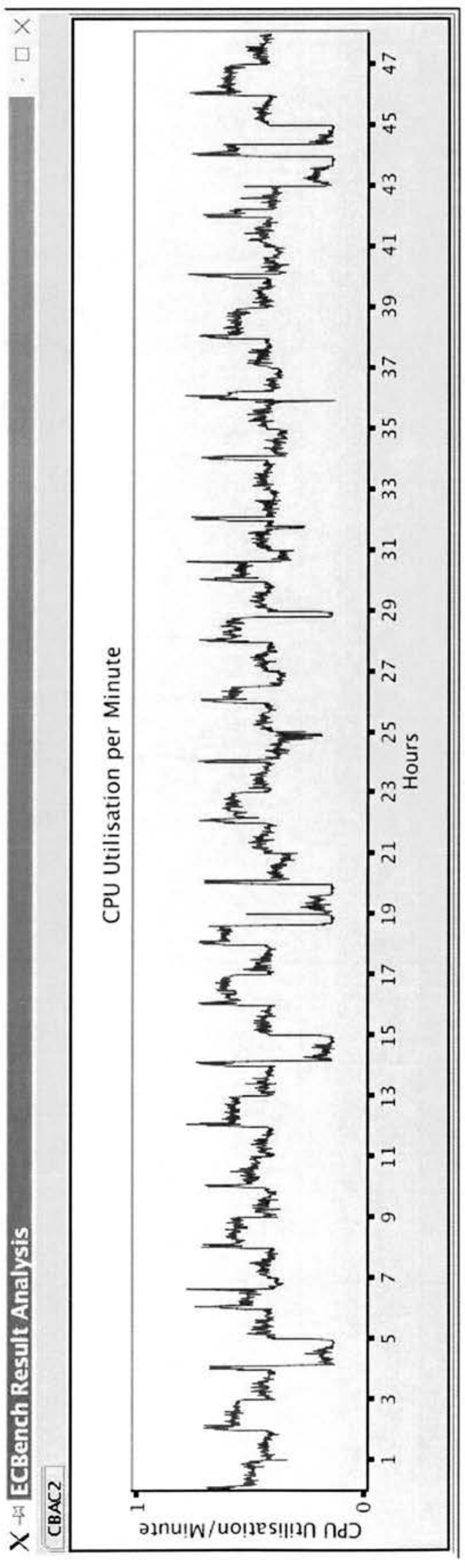


Figure 6.9: CPU Utilisation for *CBAC-disabled, CBAC1, CBAC2* and *CBAC3*



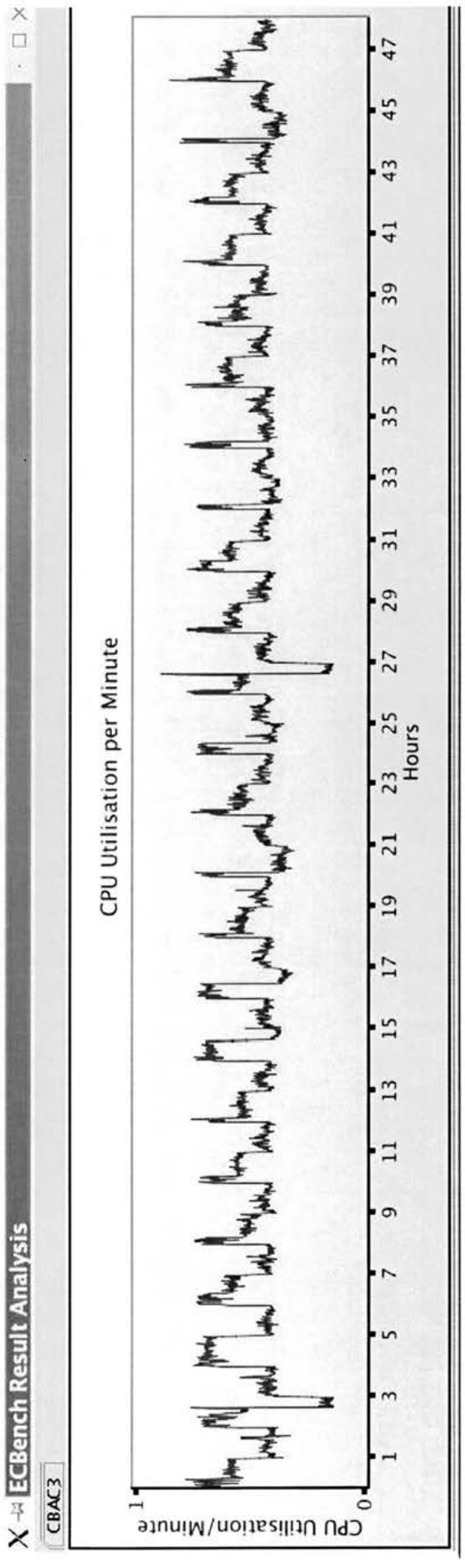


Figure 6.9: (Cont.) CPU Utilisation for *CBAC-disabled, CBAC1, CBAC2* and *CBAC3*

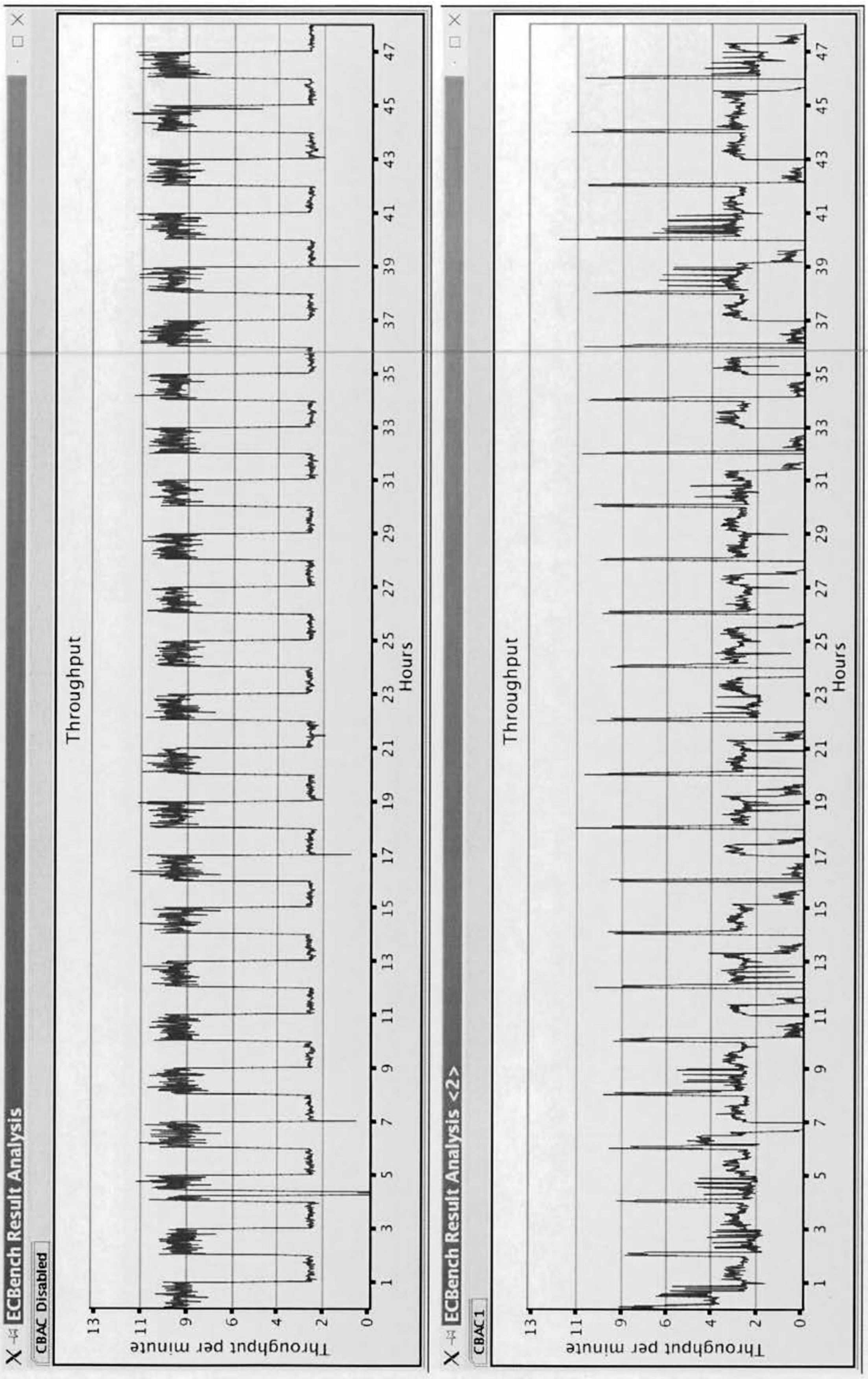
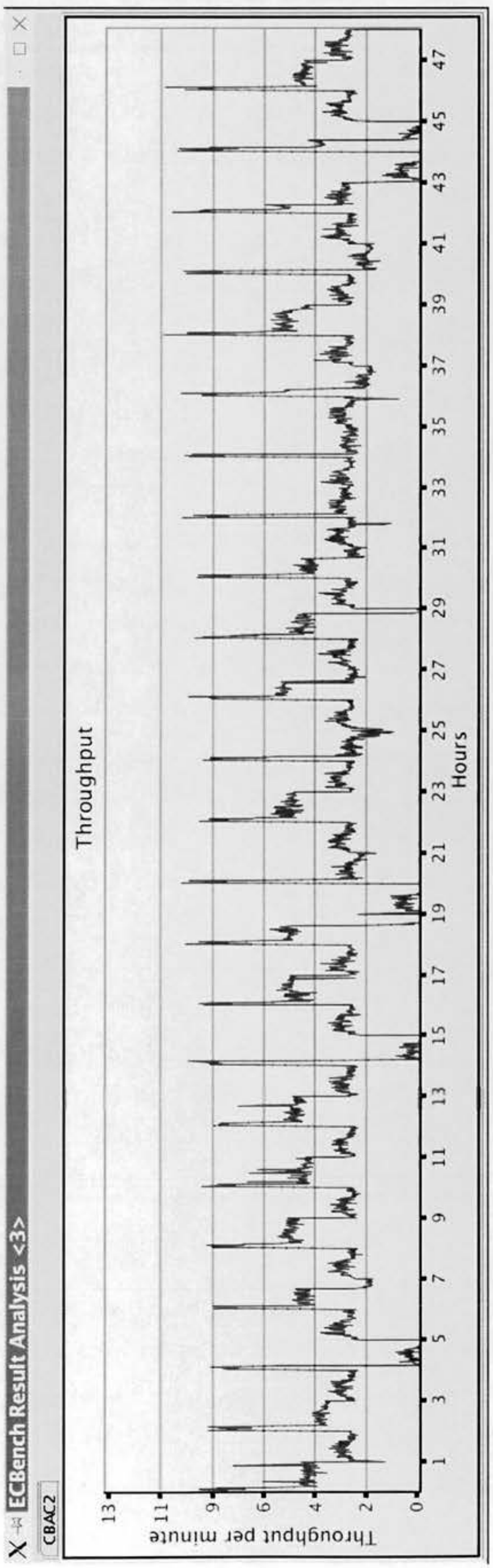


Figure 6.10: *CBAC-disabled, CBAC1, CBAC2* and *CBAC3* Throughput



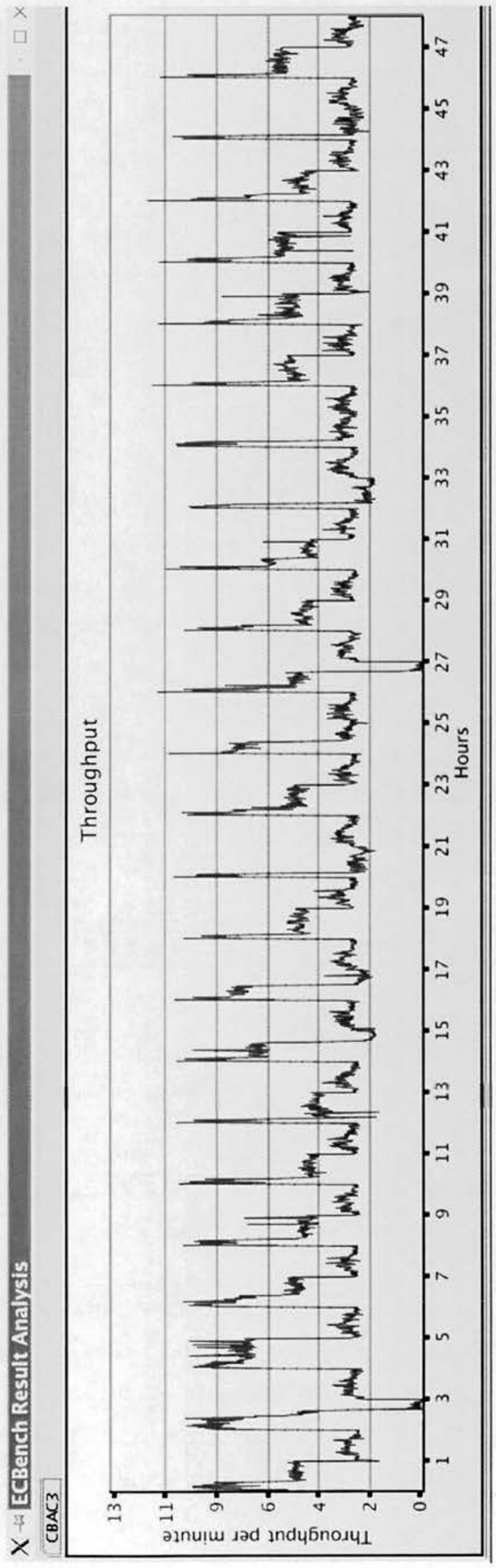


Figure 6.10: (Cont) *CBAC-disabled, CBAC1, CBAC2* and *CBAC3* Throughput

service) is one. CBAC's discount is subtracted from the basic profit. A charge is added to the basic profit in the case of a CBAC charge offer. As explained in Section 6.3.1, the upper limit of CBAC's discount/charge A_{max} is one. However, the overall CBAC charge can exceed this limit if the charging factor C is greater than one.

Figure 6.11 illustrates the profit/minute results obtained for the *CBAC-disabled*, *CBAC1*, *CBAC2* and *CBAC3* experiments. The cumulative profit per minute is depicted in Figure 6.12. The high profit at the start of high workload intervals is due to CBAC's charge offers. However, overall, it can be seen that profit in *CBAC1* drops substantially when compared with *CBAC-disabled*'s profit. One factor which contributed to this drop is the higher rate of customers accepting a discount (40%) compared to the charge acceptance rate (30%). Customer rejection of CBAC offers also contributed to the fall in profit. As discussed in Section 6.3.1, it was assumed that all rejecting customers immediately return and start a new session, which would put them further away (requesting the Home interaction) from the point where they would bring profit to the system (requesting the Buy Confirm interaction). However, the main factor which caused the drop is the no-activity intervals explained in Section 6.3.3 due to the strictness of *CBAC1* (58% threshold). During these intervals, all the workload has been scheduled for future execution and the current profit is zero. This factor becomes less significant as CBAC's threshold ϵ is raised in *CBAC2* and *CBAC3* which improves the profit dramatically.

Comparing the results of *CBAC3* and *CBAC4* gives insight into the effect of CBAC's access pattern on profit. Figure 6.13 depicts profit/minute results obtained for both experiments with the cumulative profit shown in Figure 6.12. It can be seen that a higher profit was achieved in *CBAC4*. This is due to its higher charge acceptance rate and the increased percentage of returning customers (for discount offer fulfilment).

In *CBAC5*, CBAC's charging factor C was increased to six and the profit results obtained are compared with *CBAC4* (in which C=3) as shown in Figure 6.14 and Figure 6.12. It can be seen that a substantial improvement in profit was achieved. In addition, *CBAC5*'s profit matches the profit obtained in *CBAC-disabled* and slightly exceeds it in some time intervals.

Experiment	*CBAC-disabled*	*CBAC2*
Mean Service Time (s)	0.205	0.22
CBAC Overhead (s)	0.015	

Table 6.9: CBAC Overhead Analysis at Low Workload Intervals

CBAC-Specific Interaction	*Mean Service Time (s)*
Accept Discount	0.4
Back for Offer	0.1
Fulfil Offer	0.17
Accept Charge	0.25
Reject Offer	0.26

Table 6.10: CBAC-specific Services Delay

6.3.6 CBAC Overhead

Three elements contribute to CBAC's overhead delay when compared to a CBAC-disabled setting. Firstly, the monitoring of system utilisation to determine when CBAC is applied. In the CBAC experiments performed, the server's CPU utilisation was sampled every 20 seconds, rather than per request, to minimise overhead as explained in Section 6.3.1. Such overhead can be quantified by comparing the mean service times in low workload intervals when CBAC is enabled and disabled. Table 6.9 summarises the results obtained from performing such an analysis for the *CBAC-disabled* and *CBAC2* experiments. It can be seen that a small overhead of 0.015 seconds is introduced in the CBAC experiment due to CPU sampling. At high load, when CBAC is applied, another overhead is introduced in preparing a CBAC offer for the customer. However this overhead can be neglected due to the large service time gains achieved when CBAC is applied. A more substantial CBAC overhead is introduced by the service time required to execute CBAC-specific services. Table 6.10 summarises the mean service times for such services obtained from the *CBAC2* experiment. Three possible overhead values can be introduced to the customer session given their CBAC access pattern. Accepting a discount offer, a 0.67 seconds delay is incurred due to executing services Accept Discount, Back For Offer and Fulfil Offer. This overhead is distributed between the customer's first visit to the site and the second visit when the customer returns for offer fulfilment. A total overhead of 0.25 seconds is introduced if the customer accepts CBAC's offer charge (Accept Charge service time). The Reject Offer service time represents the overhead when the customer rejects the CBAC offer.

6.3.7 CBAC Load Forecasting Effect

As described in Section 6.3.1, a one day forecasting period is used in CBAC experiments to predict future system load which is required to determine when postponed requests are to be scheduled. CBAC's load forecasting is based on the history of system load as explained in Section 5.6.3. Thus, during the first day of a CBAC experiment, load forecasting has no effect as there is no load history. At the start of the second day, such forecasting starts to contribute to CBAC scheduling. To investigate the forecasting effect, the mean service times (for Shopping Cart) in the first and second days of CBAC experiments are calculated and summarised in Table 6.11. It can be seen that there is an increase in the load forecasting effect, reflected by a reduction in the second day mean service time. This becomes more significant as CBAC's threshold ϵ is increased in *CBAC3* and *CBAC4*. In *CBAC3*, the load forecasting reduces mean service time by about 50% in the second day. *CBAC4* was less affected by load forecasting due to the increase in CBAC's charge acceptance rate. The reason for load forecasting having less impact in *CBAC1* and *CBAC2* could be due to the no-activity periods described in Section 6.3.3 which disturb load prediction.

Experiment	*Overall Mean*	*First Day Mean*	*Second Day Mean*	*Reduction*
CBAC1	0.3	0.35	0.25	28.6%
CBAC2	0.37	0.43	0.31	28%
CBAC3	0.47	0.63	0.32	49.2%
CBAC4	0.41	0.51	0.31	39.2%

Table 6.11: Service Time Analysis per Day for CBAC Experiments

6.4 Summary

The experiments reported in this chapter provide a proof-of-concept for CBAC. They show not just the feasibility of CBAC but also its great potential. Analysis of the results of the experiments has illustrated CBAC's effectiveness in providing a comprehensive enhancement framework for different Internet Commerce QoS metrics.

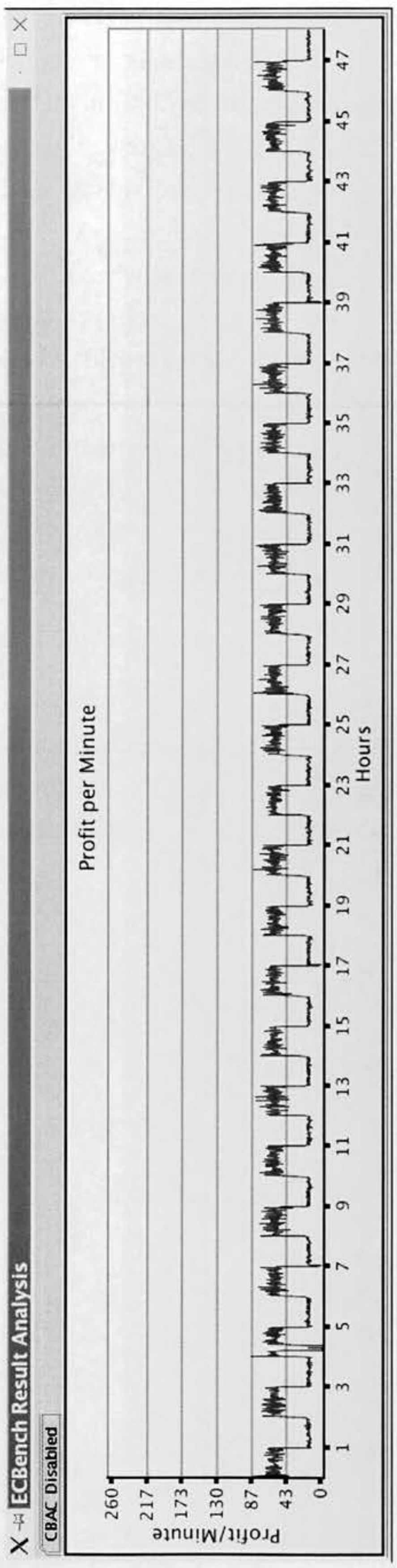
Using CBAC, a desired system utilisation can be strictly achieved providing rigorous control of system congestion. This is be accomplished without adopting a customer request rejection policy. Rather, part of the system load is distributed to future time intervals with a discount reward to improve customer satisfaction. Service times are reduced dramatically when CBAC is used, achieving a decrease by a factor of 18 compared with the performance of a CBAC-disabled system. An even further reduction in service time can be achieved by increasing the strictness of CBAC (lowering ϵ). The use of load forecasting in request scheduling enhances CBAC's performance with a significant reduction in service time of about 50% which is achieved only in the first interval forecasting is applied. The load forecasting impact is likely to increase in later intervals as load history data is accumulated.

A substantial decrease in request failure is accomplished when CBAC is activated due to the improved distribution of system load. Reasonable throughput rates are achieved and can be increased further with higher workload sizes to exploit the unused system resources exposed by CBAC.

High profit is acquired with CBAC due to its charging of customers pursuing their requests in highly congested intervals. This compensates for the profit loss caused by CBAC's discount offers. Increasing the ratio between charge and discount using CBAC's charging factor C raises the profit even further to levels above what is achieved in CBAC-disabled systems.

All of the the above improvements to different QoS metrics are achieved with minimum CBAC overhead.

The use of ECBench validates the results obtained as it provides an emulation of a real Internet Commerce system as specified in the TPC-W specification. Verfication was achieved by extensive debugging of the CBAC extensions made to ECBench to ensure correct functionality.



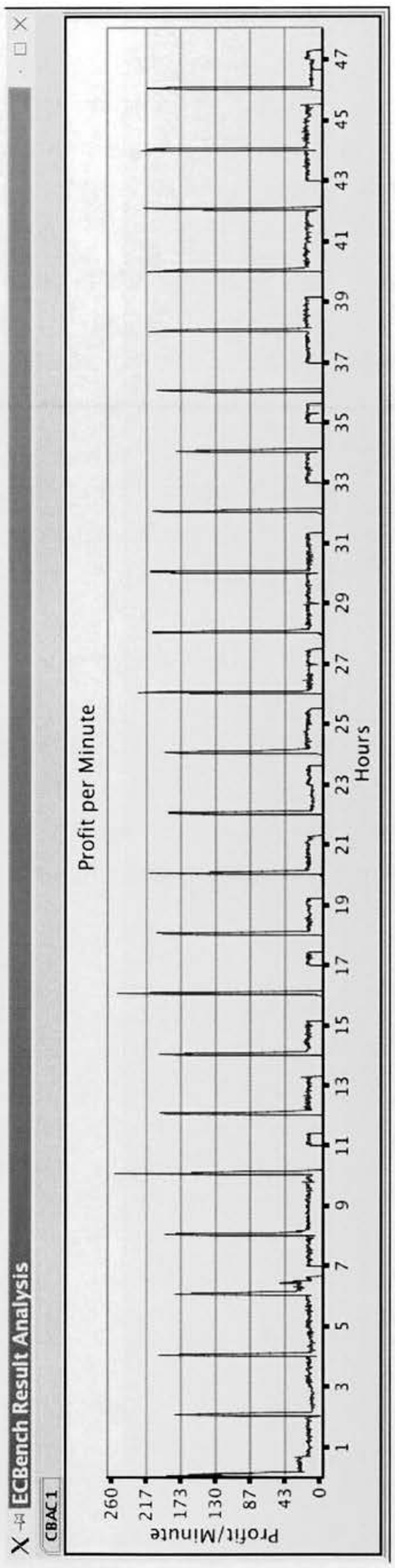
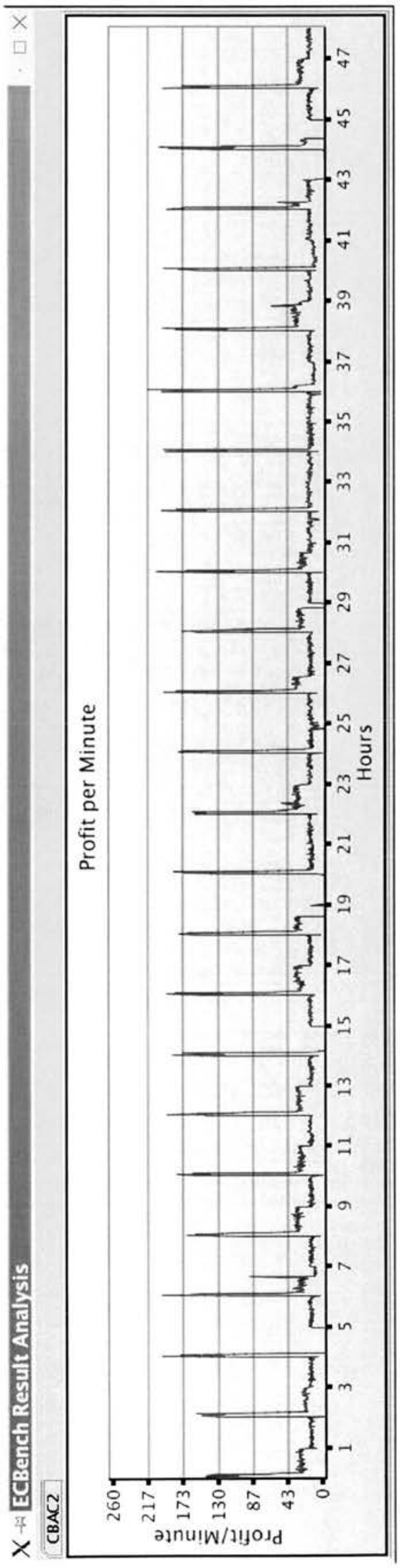


Figure 6.11: Profit Gained in *CBAC-disabled, CBAC1, CBAC2* and *CBAC3*



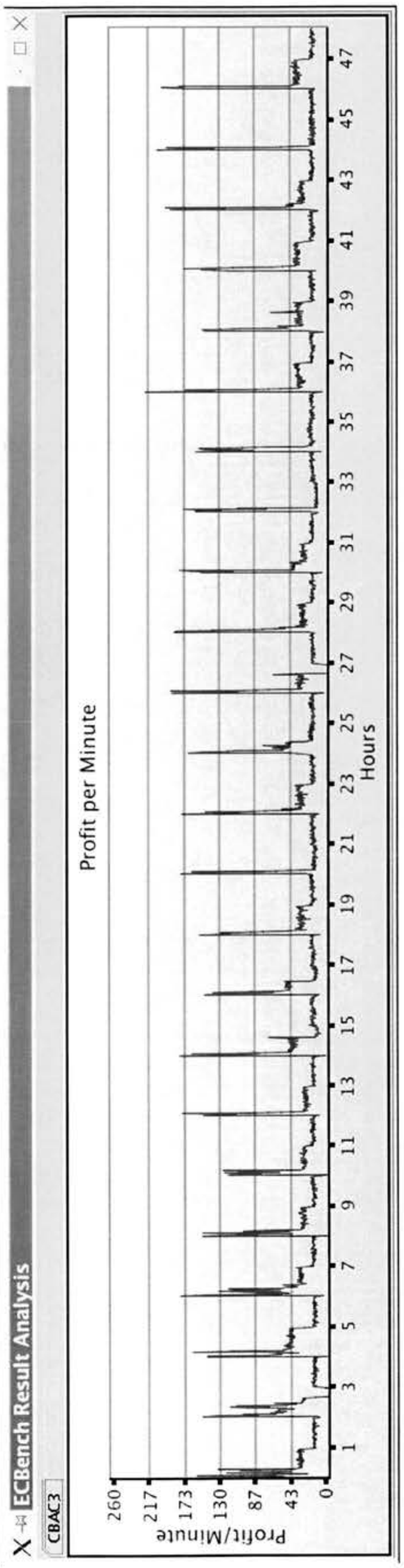


Figure 6.11: (Cont.) Profit Gained in *CBAC-disabled, CBAC1, CBAC2* and *CBAC3*

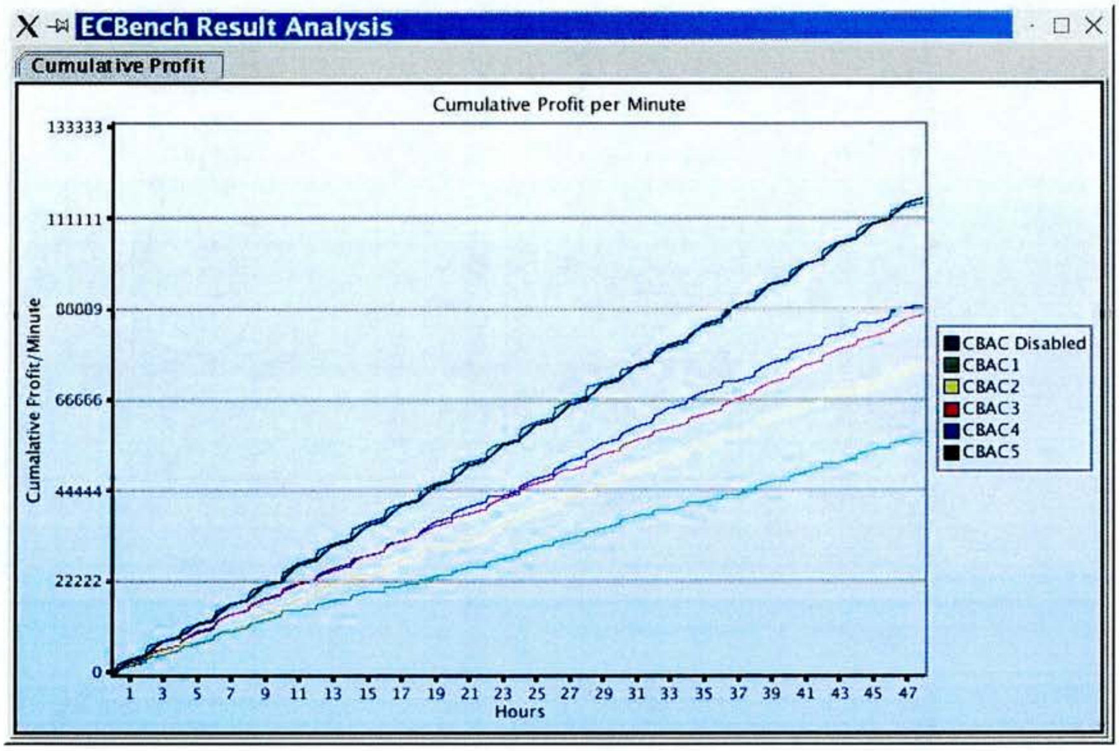


Figure 6.12: Cumulative Profit

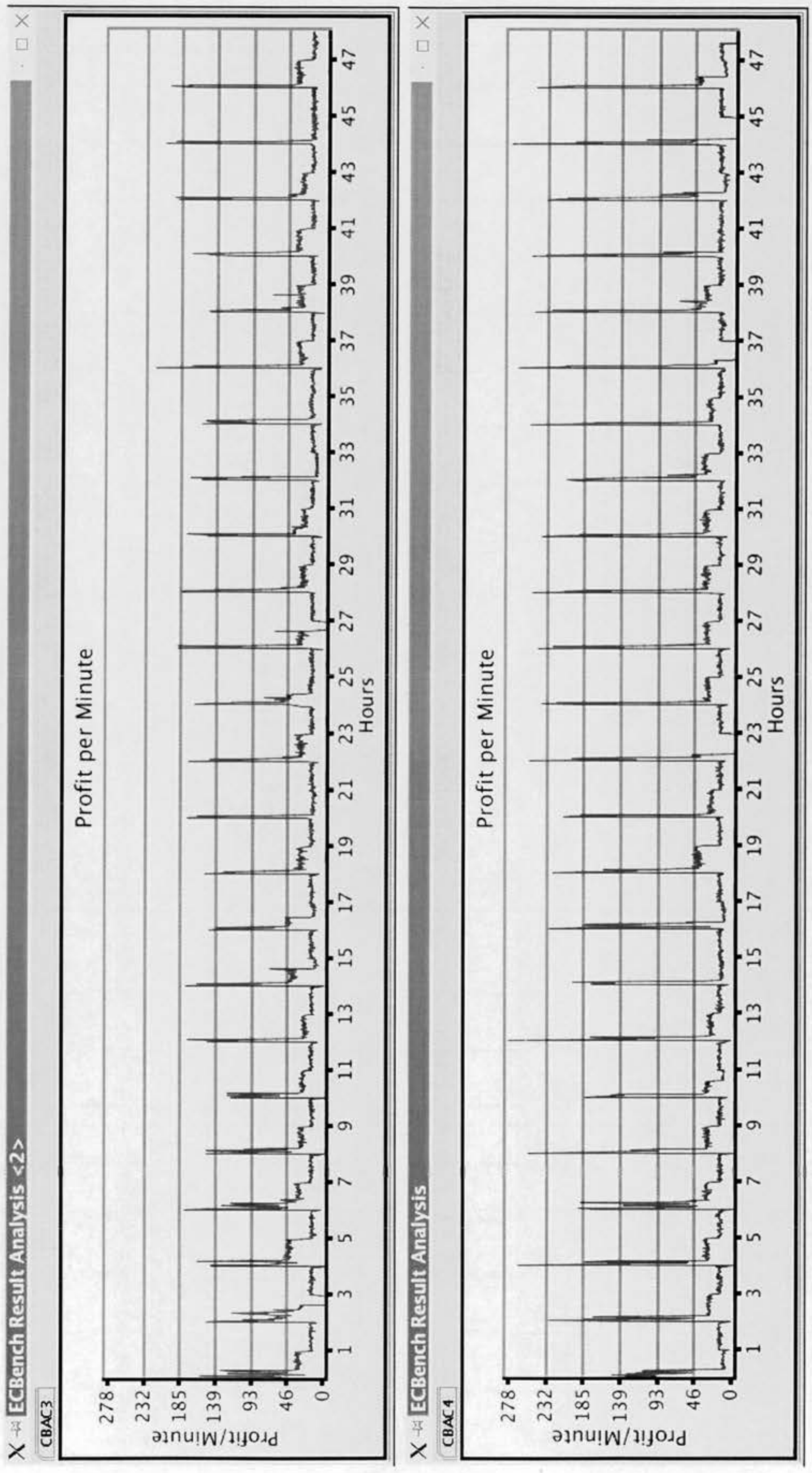


Figure 6.13: Effect of Varying CBAC Access Pattern on Profit

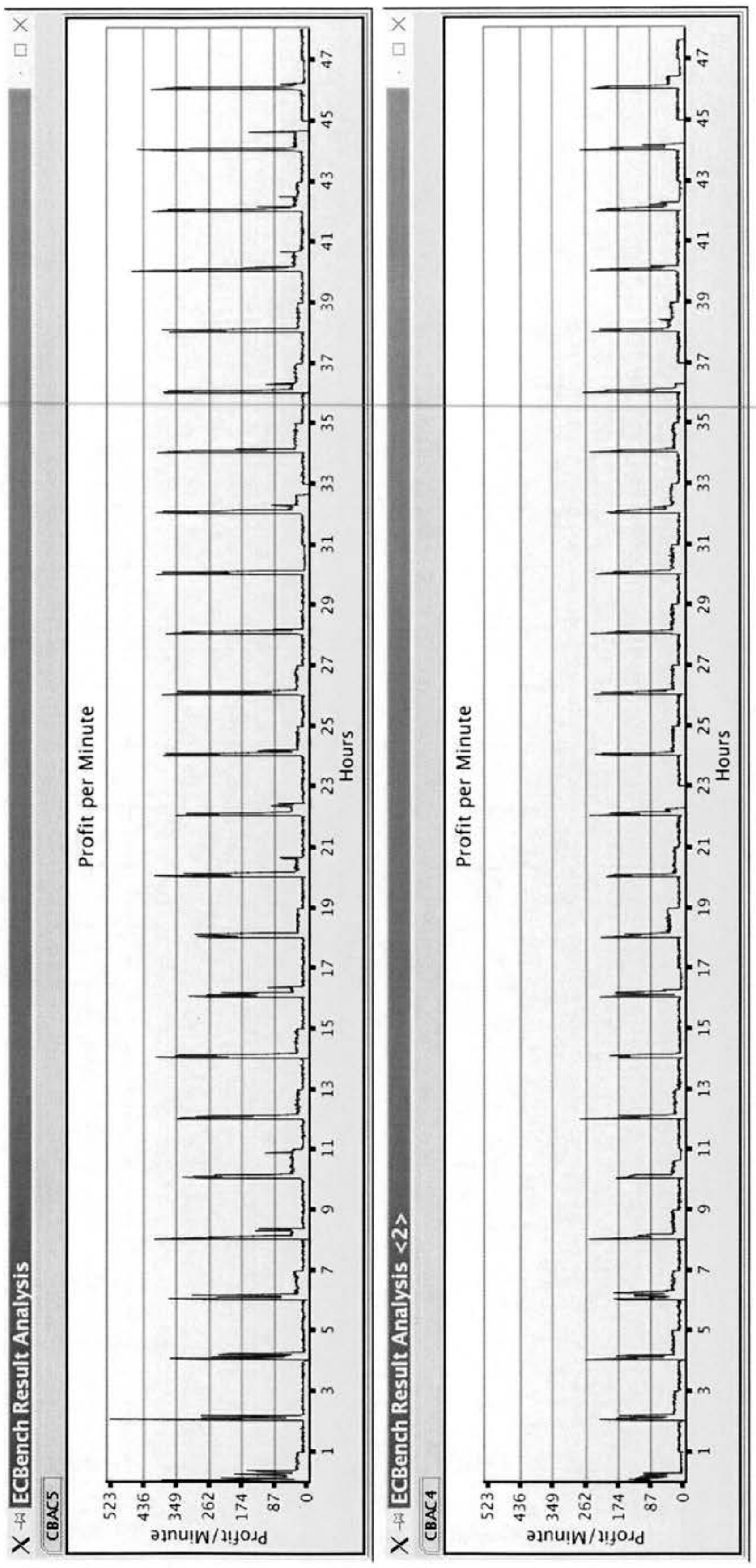


Figure 6.14: Effect of CBAC's Charge Factor on Profit

Chapter 7

Conclusions

7.1 Thesis Overview

This section provides a summary of the work presented in the thesis outlining its major contributions to the field and emphasising the work's novelty.

The first major contribution is the ECBench tool which enables the performance benchmarking of different Internet Commerce technologies and techniques. At the system design stage, ECBench can aid an Internet Commerce application developer in making QoS design decisions by choosing between alternative technologies based on their performance merits. The tool can also be used by a researcher investigating the performance of a new e-commerce technique. ECBench is a novel tool in its provision of a performance benchmarking framework which is specialised in Internet Commerce, rather than general web applications. This specialism is reflected in its use of the standard TPC-W specification providing a realistic emulation of an Internet Commerce system. ECBench also provides support for a comprehensive set of QoS metrics that are important to the performance analysis of Internet Commerce systems. It facilitates the analysis of system's profit levels, resource utilisation and throughput. Analysis of customer satisfaction can also be performed by investigating the system's service time and request failure rate. The other aspect of ECBench's novelty is that it is a general-purpose Internet Commerce tool because its modular design makes it easily extendable to benchmark new technologies and techniques. Using ECBench's experiment features, a user can design and run a set of experiments and the results obtained are analysed and presented graphically. As a case study on the use of ECBench, PHP and Java Servlets application server technologies were comparatively benchmarked using the tool.

CBAC is the second major contribution presented in this thesis. It is a novel cost-based admission control approach to enhance QoS in Internet Commerce systems. It is based on a discount-charge pricing model to encourage customers to postpone their service requests (by offering them a discount) when the system is approaching a highly loaded state. A customer is required to pay an extra charge in order to pursue their requests when the system is approaching a congested state. The pricing model is sensitive to the system's current load and navigational structure. A scheduling mechanism combined with load forecasting is used to schedule user requests in less loaded time intervals.

Analysing CBAC's performance using ECBench has shown its great potential and effectiveness in providing a comprehensive enhancement framework for different Internet Commerce QoS metrics with minimum overhead. Using CBAC, a required system utilisation can be strictly enforced by controlling system congestion without rejecting customer requests. The experiments showed that service time can be reduced dramatically with reduction by factor of 18 achievable, when compared with CBAC-disabled environments. Using one day of historical data in load forecasting, when scheduling CBAC's postponed requests, achieved a 50% reduction in service time compared with the first day CBAC was applied with no system history available. CBAC causes a substantial drop in request failures compared with CBAC-disabled systems. Throughput is maintained at reasonable levels and can be increased further at higher workload scenarios. Experiment *CBAC5* showed that CBAC's charging element and increasing the ratio between charge and discount can achieve profit levels that matches what is acquired in CBAC-disabled systems and slightly exceeds it in some time intervals.
Taken together, these experimental results show that CBAC is an innovative and effective software-based approach in maintaining all the vital Internet Commerce QoS metrics when the system is online. This is in contrast with other approaches studied and currently used. These usually try to optimise one aspect of the system's QoS and tend to target general Web applications. Being a software approach, CBAC is a financially cheaper way of load balancing than using more hardware. Customer request rejection as a way to control system congestion is another major problem with the techniques currently used and found in the literature. CBAC does not rely on such a rejection approach; rather, customers are encouraged to postpone their requests in return for a reward. In my opinion, this improves customer satisfaction and increases the probability of continued use of the system.

7.2 Future Work

The work presented in this thesis can be extended in many ways. This section provides a review of possible future research directions.

There is scope for future work on different aspects of CBAC. One possible enhancement is to add a feedback element to CBAC in which its threshold ϵ is changed dynamically based on current workload and the level at which QoS requirements are achieved. Such a feedback loop would enhance CBAC's sensitivity and reaction to variations in site traffic and thus achieve even better performance results. Another extension to CBAC's admission control pricing model is to make its discount/charge function also adaptable, not only to the current demand, but also to the business's stock supply. For example, more discount and less charge should be offered by CBAC when the stock is full of goods that must be sold in the near future. Introducing such demand-supply adaptive nature of CBAC would enhance its role as a tool to keep the competitive edge of the business.

An important issue that needs to be studied is customer's reaction to CBAC and how it is affected by different social factors. Age, gender, wealth, computer literacy and cultural background are all potential factors that could influence customer behaviour when dealing with CBAC. Considering the simpler high street store model and investigating whether customer's reaction to bargaining is similar in an Internet Commerce system could be a good starting point. A framework should be devised to reflect how factors affecting customer behaviour contribute to the setting of CBAC parameters in order to achieve optimal performance.

Another interesting direction of future work is to investigate CBAC's security. An assessment of potential security risks could be done and CBAC should be enhanced to deal with such risks. One potential security hole is a denial of service attack caused by customers rejecting CBAC offers and immediately coming back to start a new session. A mechanism is required to prevent rejected customers from returning to the system. Interesting issues arise from this such as, how long a rejected customer should be blocked from using the services offered by the system for and how the system's current and forecasted load could be used to resolve this matter.

CBAC already uses historical data for load forecasting in order to enhance the scheduling of postponed requests. Such learning from system history could be used to extend CBAC in different ways. Firstly, it could be used to learn the optimal value of ϵ as an alternative to the feedback loop discussed above. Historical data could also be used to predict a customer navigation pattern which is required for calculating time estimates when CBAC is applied. Rather than querying the customer about the number of items they are likely to purchase (as described in Section 5.5.4), system access logs could be used to derive such information. The analysis of system access logs could also be used to provide more complex scheduling strategies for CBAC's postponed requests. The times of the customer's previous visits could contribute to scheduling decisions. Scheduling policies that take into account the time-critical nature of some Internet Commerce applications should also be studied.

The scope of the experiments performed was limited by the available resources. Having more resources would allow using higher workloads to investigate how performance can be enhanced by exploiting the server's unused resources exposed by CBAC. Performing longer experiments to analyse the long term performance effect of CBAC's load forecasting could also be done. Varying CBAC's forecasting and scheduling intervals and analysing its effect on system performance could also be investigated.

CBAC could be applicable to B2B e-commerce systems. A different service model, with a lower degree of dependency level compared with Internet Commerce, would need to be used to investigate CBAC's performance impact on such systems. A different workload characterisation would also need to be used.
A portable implementation of CBAC, which can be easily integrated into e-commerce systems, should be developed. One possible way to achieve this is to implement CBAC as an Apache module which would make it feasible for CBAC to be deployed in any installation of the highly popular web server.

ECBench could be extended to provide a performance benchmarking environment for other Internet Commerce models including auction sites, portals and Electronic Markets. Further extensions could also include support for B2B e-commerce systems. Issues such as system architecture, workload characterisation and the nature of the services offered in each model would need to be studied and modelled. ECBench's GUI could be enhanced so that experiments' layout and their progress are displayed graphically to the user.

Bibliography

[1] Abu Shaaban Y. N. and Hillston J. A TPC-W-based Tool for Benchmarking E-commerce Programming Technologies. In *UK Performance Engineering Workshop, Glasgow, U.K.*, July 2002.

[2] Abu Shaaban Y. N. and Hillston J. Comparative Performance Evaluation of E-Commerce Technologies: A TPC-W-based Benchmarking Tool. In *International Conference on Analytical and Stochastic Modelling Techniques and Applications, Nottingham, U.K.*, June, 2003.

[3] TPC. http://www.tpc.org/tpcw. Web site, Transaction Processing Performance Council, 2004.

[4] Netcraft. http://news.netcraft.com/archives/web_server_survey.html. Web site, Netcraft LTD, 2004.

[5] Various Sources. Western Europe B2C E-Commerce Report. Technical report, eMarketer Inc. http://www.emarketer.com/Report.aspx?west_eur_b2c_jul04, 2004.

[6] U.S. Census Bureau. United States Department of Commerce News, http://www.census.gov/mrts/www/current.html. Web site, The U.S. Census Bureau, 2004.

[7] Whiteley D. *e-Commerce: Strategy, Technologies and Applications.* McGraw Hill, 2000.

[8] Information Society. Electronic Commerce - An Introduction. Technical report, European Commission, July 1998.

[9] Group of private sector experts on E-commerce. Sacher Report, Electronic Commerce: Opportunities and Challenges for Government, http://www.oecd.org/LongAbstract/0,2546,en_2649_37441_1893992_1_1_1_374%41,00.html. Web site, Organization for Economic Cooperation and Development (OECD), 1997.

[10] Menasce D. A. and Almeida V. A. *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning.* Prentice Hall PTR, 2000.

[11] Deitel H. M., Deitel P. J., and Steinbuhler K. *e-Business and e-Commerce for Managers.* Prentice Hall Inc., 2001.

[12] Ince D. *Developing Distributed and E-commerce Applications.* Addison Wesley, 2002.

[13] amazon.com. `www.amazon.com`. Web site, Amazon Co., 2004.

[14] etoys.com. `www.etoys.com`. Web site, etoys Co., 2004.

[15] ebay.com. `www.ebay.com`. Web site, ebay Co., 2004.

[16] yahoo.com. `www.yahoo.com`. Web site, Yahoo Co., 2004.

[17] Jain R. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, Inc., 1991.

[18] Freier A., Karlton P., and Kocher P. The SSL Protocol, `http://wp.netscape.com/eng/ssl3/draft302.txt`. Web site, Netscape Communications, 1996.

[19] The SET Security Protocol, `www.setco.org`. Web site.

[20] Kristol D. and Montulli L. HTTP State Management Mechanism, `http://www.ietf.org/rfc/rfc2109.txt`. Web site, Network Working Group, IETF, 1997.

[21] Arlitt M. F. and Williamson C. L. Internet Web Servers: Workload Characterization and Performance Implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, Oct. 1997.

[22] Arlitt M. F. and Williamson C. L. Web Server Workload Characterization: The Search for Invariants. In *ACM SIGMETRICS on Measurement and Modeling of Computer Systems*, pages 126–137, Philadelphia, U.S., May 1996.

[23] Crovella M. E. and Bestavros A. Self-similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, Dec. 1997.

[24] Almeida V., Crovella M., Bestavros A., and Oliveira. Characterizing Reference Locality in the WWW. In *IEEE/ACM International Conference on Parallel and Distributed Systems*, pages 92–107, Florida, U.S., Dec 1996.

[25] Huberman B. A., Pirolli P. L. T., Pitkow J. E., and Lukose R. M. Strong Regularities in World Wide Web Surfing. *Science*, 280(3):95–97, April 1998.

[26] Schechter S., Krishnan M., and Smith M. D. Using Path Profiles to Predict HTTP Requests. *ACM Computer Networks and ISDN Systems*, 30(1-7):457–467, 1998.

[27] Padmanabhan V. N. and Mogul J. C. Using Predictive Prefetching to Improve World Wide Web Latency. *ACM SIGCOMM Computer Communication Review*, 26(3):22–36, 1996.

[28] Bestavros A. Using Speculation to Reduce Server Load and Service Time on the WWW. In *ACM International Conference on Information and Knowledge Management*, pages 403–410, Baltimore, U.S., Nov. 1995.

[29] Pitkow J. and Pirolli P. Mining Longest Repeating Subsequences to Predict World Wide Web Surfing. In *USENIX Symposium on Internet Technologies and Systems*, pages 11–14, Colorado, U.S., Oct. 1999.

[30] Deshpande M. and Karypis G. Selective Markov Models for Predicting Webpage Accesses. *ACM Transactions on Internet Technology*, 4(2):163–184, 2004.

[31] Su Z., Yang Q., Lu Y., and Zhang H. WhatNext: A Prediction System for Web Requests using N-gram Sequence Models. In *International Conference on Web Information Systems Engineering*, pages 200–207, Hong Kong, 2000.

[32] Li I. T. Y., Yang Q., and Wang K. Classification Pruning for Web-request Prediction. In *International World Wide Web Conference (Poster)*, Hong Kong, 2001.

[33] Arlitt M., Krishnamurthy D., and Rolia J. Characterizing the Scalability of a Large Web-based Shopping System. *ACM Transactions on Internet Technology*, 1(1):44–69, 2001.

[34] Menasce D., Almeida V., and Riedi R. In Search of Invariants for E-business Workloads. In *ACM Electronic Commerce Conference*, pages 56–65, Minneapolis, U.S., Oct. 2000.

[35] Almeida V., Menasce D., Riedi R., Peligrinelli F., Fonseca R., and Meira Jr. W. Analyzing Robot Behavior in E-business Sites. In *ACM SIGMETRICS on Measurement and Modeling of Computer Systems*, pages 338–339, Cambridge, MA, U.S., 2001.

[36] Menasce D. and Almeida V. A Methodology for Workload Characterization of E-commerce Sites. In *ACM Electronic Commerce Conference*, pages 119–129, Denver U.S., 1999.

[37] Wang Q., Makaroff D. J., and Edwards H. K. Characterizing Customer Groups for an E-commerce Website. In *ACM Electronic Commerce Conference*, pages 218–227, New York, U.S., May 2004.

[38] Yun C. and Chen M. Mining Web Transaction Patterns in an Electronic Commerce Environment. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, pages 216–219, Kyoto Japan, 2000. Springer-Verlag.

[39] Vallamkondu S. and Gruenwald L. Integrating Purchase Patterns and Traversal Patterns to Predict HTTP Requests in E-commerce Sites. In *IEEE International Conference on E-commerce*, pages 256–263, California, U.S., 2003.

[40] SPECweb99 Whitepaper, `http://www.specbench.org/osg/web99/docs/whitepaper.html`. Web site, Standard Performance Evaluation Corporation (SPEC), July, 2000.

[41] WebStone 2.x Benchmark Description, `http://www.mindcraft.com/webstone/ws201-descr.html`. Web site, Mindcraft Inc., 1998.

[42] Banga G. and Druschel P. Measuring the Capacity of a Web Server. In *USENIX Symposium on Internet Technologies and Systems*, pages 61–72, California, U.S., Dec. 1997.

[43] Mosberger D. and Jin T. httperf - A Tool for Measuring Web Server Performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37, 1998.

[44] Barford P. and Crovella M. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *ACM SIGMETRICS on Measurement and Modeling of Computer Systems*, pages 151–160, Wisconsin, U.S., 1998.

[45] Kant K., Tewari V., and Iyer R. Geist: A Generator for E-commerce & Internet Server Traffic. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 49–56, Arizona, U.S., Nov. 2001.

[46] e-TEST suite. `http://www.empirix.com`. Web site, Empirix, 2004.

[47] The Source for Java Developers, `http://java.sun.com`. Web site, Sun Mircosystems Inc., 2004.

[48] Simas J. Chart2d, `http://chart2d.sourceforge.net`. Web site, Free Software Foundation, Inc., 1999.

[49] Apache. `http://www.apache.org`. Web site, The Apache Software Foundation, 2004.

[50] MySQL. `http://www.mysql.com`. Web site, MySQLAB, 2004.

[51] S. Gundavaram. *CGI Programming on the World Wide Web.* O'Reilly & Associates, 1996.

[52] Active Server Pages. `http://msdn.microsoft.com/library/psdk/iisref/aspguide.htm`. Web site, Microsoft Corporation, 2004.

[53] ColdFusion. ColdFusion Developer Center, `http://www.macromedia.com/devnet/mx/coldfusion/?promoid=home_dev_cf_082%403`. Web site, Macromedia, Inc., 2004.

[54] Perry B. *Java Servlet & JSP Cookbook.* O'Reilly & Associates, 2004.

[55] Hunter J. and Crawford W. *Java Servlet Programming.* O'Reilly & Associates, 2nd Edition, 2001.

[56] Lerdorf R. and Tatroe K. *Programming PHP.* O'Reilly & Associates, 2002.

[57] Welling L. and Thomson L. *PHP and MySQL Web Development.* SAMS, 2nd Edition, 2003.

[58] PHP Usage. `http://www.php.net/usage.php`. Web site, The PHP Group, 2004.

[59] Enterprise Edition (J2EE) Java 2 Platform. `http://java.sun.com/j2ee`. Web site, Sun Microsystems Inc., 2004.

[60] PHP. `http://www.php.net`. Web site, The PHP Group, 2004.

[61] Mazzocchi S. and Fumagalli P. Advanced Apache Jserv Techniques. In *ApacheCon, San Francisco*, 1998.

[62] Red Hat Linux 9: Red Hat Linux Reference Guide. `http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ref-guide/`. Web site, Red Hat Inc., 2003.

[63] Servlet Problem. `http://www.mail-archive.com/java-apache-users@list.working-dogs.com/msg%07417.html`. Web site, java-apache-users Mailing list, 2000.

[64] Cardellini V., Casalicchio E., Colajanni M., and Yu P. S. The State of the Art in Locally Distributed Web-server Systems. *ACM Computer Surveys*, 34(2):263–311, June 2002.

[65] Katz E. D., Butler M., and McGrath R. A Scalable HTTP Server: The NCSA Prototype. *Computer Networks and ISDN Systems*, 27(2):155–164, 1994.

[66] Colajanni M. and Yu P. S. Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers. In *IEEE International Distributed Computing Systems Conference*, pages 295–302, Amsterdam, The Netherlands, May 1998.

[67] Colajanni M. and Yu P. S. Adaptive TTL Schemes for Load Balancing of Distributed Web Servers. *ACM SIGMETRICS Performance Evaluation Review*, 25(2):36–42, 1997.

[68] Cardellini V., Colajanni M., and Yu P. S. Redirection Algorithms for Load Sharing in Distributed Web-server Systems. In *IEEE International Conference on Distributed Computing Systems*, pages 528–535, Texas, U.S., May 1999.

[69] Aversa L. and Bestavros A. Load Balancing a Cluster of Web Servers Using Distributed Packet Rewriting. In *IEEE International Performance, Computing and Communications Conference*, pages 24–29, Boston, U.S., 2000.

[70] Yang C. and Luo M. A Content Placement and Management System for Distributed Web-server Systems. In *IEEE Distributed Computing Systems*, pages 691–698, Taipei, Taiwan, April 2000.

[71] Zhang X., Barrientos M., Chen J. B., and Seltzer M. HACC: An Architecture for Cluster-based Web Servers. In *USENIX Windows NT Symposium*, pages 155–164, Seatle U.S., July 1999.

[72] Luo M. and Yang C. Constructing Zero-loss Web Services. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1781–1790, Alaska, U.S., 2001.

[73] Cherkasova L. and Karlsson M. Scalable Web Server Cluster Design with Workload-Aware Request Distribution Strategy WARD. In *IEEE International Workshop on Advanced Issues of E-commerce and Web-based Information Systems*, page 212, California, U.S., June 2001.

[74] Zhu H., Smith B., and Yang T. Scheduling Optimization for Resource-intensive Web Requests on Server Clusters. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 13–22, Saint Malo, France, 1999.

[75] Pierre G., Steen M. V., and Tanenbaum A. S. Dynamically Selecting Optimal Distribution Strategies for Web Documents. *IEEE Transactions on Computers*, 51(6):637–651, 2001.

[76] Aron M., Sanders D., and Druschel P. Scalable Content-aware Request Distribution in Cluster-based Network Servers. In *USENIX Annual Technical Conference*, pages 323–336, California, U.S., June 2000.

[77] Dias D. M., Kish W., Mukherjee R., and Tewari R. A Scalable and Highly Available Web Server. In *IEEE International Computer Conference: Technologies for the Information Superhighway*, pages 85–92, California, U.S., 1996.

[78] Chen H. and Iyengar A. A Tiered System for Serving Differentiated Content. *World Wide Web Journal*, 6(4):331–352, 2003.

[79] Casalicchio E. and Colajanni M. A Client-aware Dispatching Algorithm for Web Clusters Providing Multiple Services. In *ACM World Wide Web Conference*, pages 535–544, Hong Kong, May 2001.

[80] Ciardo G., Riska A., and Smirni E. EQUILOAD: a Load Balancing Policy for Clustered Web Servers. *Performance Evaluation*, 46(2-3):101–124, Oct 2001.

[81] Sayal M., Breitbart Y., Scheuermann P., and Vingralek R. Selection Algorithms for Replicated Web Servers. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):44–50, 1998.

[82] Shaikh A., Tewari R., and Agrawal M. On the Effectiveness of DNS-based Server Selection. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1801–1810, Alaska, U.S., 2001.

[83] Yoshikawa C., Chun B., Eastham P., Vahdat A., Anderson T., and Culler D. Using Smart Clients to Build Scalable Services. In *USENIX Technical Conference*, pages 105–118, California, U.S., Jan 1997.

[84] Vingralek R., Breitbart Y., Sayal M., and Scheuermann P. Web++: A System For Fast and Reliable Web Service. In *USENIX Annual Technical Conference*, pages 171–184, California, U.S., 1999.

[85] Fei Z., Bhattacharjee S., Zegura E. W., and Ammar M. H. A Novel Server Selection Technique for Improving the Response Time of a Replicated Service. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 783–791, California, U.S., 1998.

[86] Carter R. L. and Crovella M. E. Dynamic Server Selection Using Bandwidth Probing on Wide-area Networks. Technical Report BU-CS-96-007, Computer Science Department, Boston University, 1996.

[87] Yu T. and Lin K. The Design of QoS Broker Algorithms for QoS-capable Web Services. In *IEEE International Conference on e-Technology, e-Commerce and e-Service*, pages 17–24, Taipei, Taiwan, 2004.

[88] Vahdat A., Anderson T., Dahlin M., Belani E., and Culler D. WebOS: Operating System Services for Wide Area Applications. In *IEEE Symposium on High Performance Distributed Computing*, pages 52–63, Chicago, U.S., July 1998.

[89] Pierre G. and Steen M. V. Globule: A Platform for Self-Replicating Web Documents. In *International Conference on Protocols for Multimedia Systems*, pages 1–11, Enschede, The Netherlands, 2001. Springer-Verlag.

[90] Li Q. and Moon B. Distributed Cooperative Apache Web Server. In *ACM International World Wide Web Conference*, pages 555–564, Hong Kong, 2001.

[91] Abrams M., Standridge C. R., Abdulla G., Williams S., and Fox E. A. Caching Proxies: Limitations and Potentials. In *International World Wide Web Conference*, Boston U.S., Dec. 1995.

[92] Kroeger T. M. and Long D. D. E. Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. In *USENIX Symposium on Internet Technologies and Systems*, pages 13–22, California, U.S., Dec. 1997.

[93] Markatos E. P. and Chronaki C. E. A Top-10 Approach to Prefetching on the Web. In *Annual Conference of the Internet Society*, Geneva, Switzerland, July 1998.

[94] Chinen K. and Yamaguchi S. An Interactive Prefetching Proxy Server for Improvement of WWW Latency. In *Annual Conference of the Internet Society*, Kuala Lumpur, Malaysia, June 1997.

[95] Fan L., Cao P., and Jacobson Q. Web Prefetching Between Low-bandwidth Clients and Proxies: Potential and Performance. In *ACM SIGMETRICS International Conference on Measurement and Modelling of Computer Systems*, pages 178–187, Atlanta, U.S., May 1999.

[96] Cao P. and Irani S. Cost-aware WWW Proxy Caching Algorithms. In *USENIX Symposium on Internet Technology and Systems*, pages 193–206, California, U.S., Dec. 1997.

[97] Scheuermann P., Shim J., and Vingralek R. A Case for Delay-conscious Caching of Web Documents. *Computer Networks and ISDN Systems*, 29(8-13):997–1005, 1997.

[98] Shim J., Scheuermann P., and Vingralek R. A Unified Algorithm for Cache Replacement and Consistency in Web Proxy Servers. In *International Workshop on the the World Wide Web and Databases*, pages 1–13, Valencia, Spain, 1998. Springer-Verlag.

[99] Wooster R. P. and Abrams M. Proxy Caching that Estimates Page Load Delays. In *International World Wide Web Conference*, pages 977–986, California, U.S., 1997. Elsevier Science Publishers Ltd.

[100] Niclausse N., Liu Z., and Nain P. A New and Efficient Caching Policy for the World Wide Web. In *Workshop on Internet Server Performance*, pages 119–128, Madison, U.S., June 1998.

[101] Tewari R., Vin H. M., Dan A., and Sitaram D. Resource-based Caching for Web Servers. In *SPIE/ACM Conference on Multimedia and Networking*, pages 191–204, California, U.S., Jan. 1998.

[102] Fan L., Cao P., Almeida J., and Broder A. Z. Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol. In *ACM SIGCOMM Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*, pages 254–265, Vancouver, Canada, 1998.

[103] Rabinovich M., Chase J., and Gadde S. Not All Hits Are Created Equal: Cooperative Proxy Caching Over a Wide-area Network. *Computer Networks and ISDN Systems*, 30(22-23):2253–2259, Nov. 1998.

[104] Zhang L., Floyd S., and Jacobson V. Adaptive Web Caching. In *NLANR Web Cache Workshop*, Colorado, U.S., June 1997.

[105] Song J., Iyengar A., Levy-Abegnoli E., and Dias D. Architecture of a Web Server Accelerator. *Computer Networks*, 28(1):75–97, 2002.

[106] Holmedahl V., Smith B., and Yang T. Cooperative Caching of Dynamic Content on a Distributed Web Server. In *IEEE International Symposium on High Performance Distributed Computing*, pages 243–250, Chicago, U.S., July 1998.

[107] Zhu H. and Yang T. Class-based Cache Management for Dynamic Web Content. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1215–1224, Alaska, U.S., April 2001.

[108] Challenger J. R., Dantzig P., Iyengar A., Squillante M. S., and Zhang L. Efficiently Serving Dynamic Data at Highly Accessed Web Sites. *IEEE/ACM Transactions on Networking*, 12(2):233–246, April 2004.

[109] Ramaswamy L., Iyengar A., Liu L., and Douglis F. Automatic Detection of Fragments in Dynamically Generated Web Pages. In *ACM World Wide Web Conference*, pages 443–454, New York, U.S., May 2004.

[110] Datta A., Dutta K., Thomas H., VanderMeer D., Suresha, and Ramamritham K. Proxy-based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation. In *ACM SIGMOD International Conference on Management of Data*, pages 97–108, Wisconsin, U.S., June 2002.

[111] Cao P., Zhang J., and Beach K. Active Cache: Caching Dynamic Contents on the Web. *Distributed Systems Engineering*, 6(1):43–50, 1999.

[112] Meira Jr. W., Menasce D., Almeida V., and Fonseca R. E-representative: a Scalability Scheme for E-commerce. In *IEEE International Workshop on Advanced Issues of E-commerce and Web-based Information Systems*, page 168, California, U.S., 2000.

[113] Luo Q. and Naughton J. F. Form-based Proxy Caching for Database-backed Web Sites. In *International Conference on Very Large DataBases*, pages 191–200, Roma, Italy, Sept. 2001. Morgan Kaufmann.

[114] Yuan C., Chen Y., and Zhang Z. Evaluation of Edge Caching/Offloading for Dynamic Content Delivery. In *ACM World Wide Web Conference*, pages 461–471, Budapest Hungary, May 2003.

[115] Abdelzaher T. F. and Bhatti N. Web Server QoS Management by Adaptive Content Delivery. In *International Workshop on Quality of Service*, London, U.K., 1999.

[116] Muntean C. H., McManis J., Murphy J., and Murphy L. An Adaptive Web Server Application. In *International World Wide Conference (Poster)*, Hawaii, U.S., May 2002.

[117] Krishnamurthy B., Zhang Y., Wills C. E., and Vishwanath K. Design, Implementation, and Evaluation of a Client Characterization Driven Web Server. In *ACM International World Wide Web Conference*, pages 138–147, Budapest, Hungary, 2003.

[118] Chandra S., Ellis C. S., and Vahdat A. Differentiated Multimedia Services Using Quality Aware Transcoding. In *Annual Joint Conference of the IEEE Computer And Communications Societies*, pages 961–969, Tel Aviv, Israel, 2000.

[119] Wills C. E., Mikhailov M., and Shang H. N for the Price of 1: Bundling Web Objects for More Efficient Content Delivery. In *ACM International World Wide Web Conference*, pages 257–265, Hong Kong, May 2001.

[120] Fox A. and Brewer E. A. Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation. In *International World Wide Web Conference on Computer Networks and ISDN Systems*, pages 1445–1456, Paris, France, 1996. Elsevier Science Publishers B. V.

[121] Mogul J. C. Potential Benefits of Delta Encoding and Data Compression for HTTP. In *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 181–194, Cannes, France, 1997.

[122] Mohapatra P. and Chen Huamin. A Framework for Managing QoS and Improving Performance of Dynamic Web Content. In *IEEE GLOBECOM*, Texas, U.S., Nov. 2001.

[123] amazon.com. Method and system for placing a purchase order via a communications network. Technical report, United States Patent Number 5,960,411, Sept. 1999.

[124] Abdelzaher T. F. and Shin K. G. Performance Guarantees for Web Server End-systems: A Control-theoretical Approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, Jan 2002.

[125] Robertsson A., Wittenmark B., and Kihl M. Analysis and Design of Admission Control in Web-server Systems. In *American Control Conference*, page 254, Colorado, U.S., 2003.

[126] Diao Y., Gandhi N., and Hellerstein J. L. Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics with Application to the Apache Web Server. In *IEEE/IFIP Network Operations and Management Symposium*, volume 8, pages 219–234, Florence Italy, April 2002.

[127] Abdelzaher T., Lu Ying, Zhang R., and Henriksson D. Practical Application of Control Theory to Web Services. In *American Control Conference*, Boston, U.S., June 2004.

[128] Lu Y., Abdelzaher T., and Lu C. Feedback Control with Queueing-theoretic Prediction for Relative Delay Guarantees in Web Servers. In *IEEE Real-time and Embedded Technology and Applications Symposium*, page 208, Toronto, Canada, May 2003.

[129] Henriksson D., Lu Y., and Abdelzaher T. Improved Prediction for Web Server Delay Control. In *Euromicro Conference on Real-time Systems*, pages 61–68, Catania Italy, July 2004.

[130] Menasce D. A., Barbara D., and Dodge R. Preserving QoS of E-commerce Sites Through Self-tuning: A Performance Model Approach. In *ACM Electronic Commerce Conference*, pages 224–234, Florida, U.S., Oct. 2001.

[131] Iyer R., Tewari V., and Kant K. Overload Control Mechanisms for Web Servers. In *Performance and QoS of Next Generation Networks*, pages 225–244, Nagoya, Japan, Nov 2000.

[132] Cherkasova L. and Phaal P. Session Based Admission Control: A Mechanism for Improving Performance of Commercial Web Sites. *IEEE Transactions on Computers*, 51(6):669–685, June 2002.

[133] Chen H. and Mohapatra P. Session-based Overload Control in QoS-aware Web Servers. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, New York, U.S., June 2002.

[134] Bhatti N. and Friedrich R. Web Server Support for Tiered Services. *IEEE Network*, 13:64–71, Sept-Oct 1999.

[135] Voigt T., Tewari R., Freimuth D., and Mehra A. Kernel Mechanisms for Service Differentiation in Overloaded Web Servers. In *USENIX Annual Technical Conference*, pages 189–202, Boston, U.S., June 2001.

[136] Jamjoom H., Reumann J., and Shin K. G. QGuard: Protecting Internet Servers from Overload. Technical Report CSE-TR-427-00, Department of Electrical Engineering and Computer Science, University of Michigan, 2000.

[137] Chen X. and Mohapatra P. Providing Differentiated Service from an Internet Server. In *IEEE International Conference on Computer Communications and Networks*, Boston, U.S., 1999.

[138] Vasiliou N. and Lutfiyya H. Providing A Differentiated Quality of Service in a World Wide Web Server. *ACM Performance Evaluation Review*, 28(2):22–28, 2000.

[139] Li K. and Jamin S. A Measurement-based Admission-controlled Web Server. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 651–659, Tel Aviv, Israel, 2000.

[140] Lee S. C. M., Lui J. C. S., and Yau D. K. Y. Admission Control and Dynamic Adaptation for a Proportional-delay DiffServ-enabled Web Server. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 172–182, California, U.S., June 2002.

[141] Kanodia V. and Knightly E. W. Ensuring Latency Targets in Multiclass Web Servers. *IEEE Transactions on Parallel and Distributed Systems*, 14(1):84–93, 2003.

[142] Chen X., Mohapatra P., and Chen H. An Admission Control Scheme for Predictable Server Response Time for Web Accesses. In *ACM International World Wide Web Conference*, pages 545–554, Hong Kong, May 2001.

[143] Crovella M. E., Frangioso R., and Harchol-Balter M. Connection Scheduling in Web Servers. In *USENIX Symposium on Internet Technologies and Systems*, Colorado, U.S., Oct 1999.

[144] Schroeder B. and Harchol-Balter M. Web Servers Under Overload: How Scheduling Can Help. Technical Report CMU-CS-020143, Carnegie Mellon University, Pittsburgh, May 2002.

[145] Cherkasova L. Scheduling Startegy to Improve Response Time for Web Applications. In *International Conference and Exhibition on High Performance Computing and Networking*, pages 305–314, Amsterdam, The Netherlands, April 1998. Springer-Verlag.

[146] Welsh M., Culler D., and Brewer E. SEDA: an Architecture for Well-conditioned, Scalable Internet Services. In *ACM Symposium on Operating Systems Principles*, pages 230–243, Alberta, Canada, Oct 2001.

[147] Welsh M. and Culler D. Adaptive Overload Control for Busy Internet Servers. In *USENIX Symposium on Internet Technologies and Systems*, Seattle, U.S., March 2003.

[148] Carlstrom J. and Rom R. Application-aware admission control and scheduling in web servers. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 506–515, New York, U.S., 2002.

[149] Verma A. and Ghosal S. On Admission Control for Profit Maximization of Networked Service Providers. In *ACM International World Wide Web Conference*, pages 128–137, Budapest, Hungary, May 2003.

[150] Abdelzaher T. F. and Shin K. G. QoS Provisioning with qContracts in Web and Multimedia Servers. In *IEEE Real-time Systems Symposium*, pages 44–53, Phoenix, U.S., 1999.

[151] Abdelzaher T. F., Shin K. G., and Bhatti N. User-level QoS-adaptive Resource Management in Server End-systems. *IEEE Transactions on Computers*, 52(5):678–685, May 2003.

[152] Chen I. and Li S. A Cost-based Admission Control Algorithm for Handling Mixed Workloads in Multimedia Server Systems. In *IEEE International Conference on Parallel and Distributed Systems*, pages 543–548, KyongJu City, Korea, 2001.

[153] Vin H. M., Goyal P., Goyal A., and Goyal A. A Statistical Admission Control Algorithm for Multimedia Servers. In *ACM International Multimedia Conference*, pages 33–40, San Francisco, U.S., Oct 1994.

[154] Lee W. and Sabata B. Admission Control and QoS Negotiations for Soft-real Time Applications. In *IEEE International Conference on Multimedia Computing and Systems*, volume 1, pages 147–152, Florence Italy, June 1999.

[155] Zhu H., Tang H., and Yang T. Demand-driven Service Differentiation in Cluster-based Network Servers. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 679–688, Alaska, U.S., 2001.

[156] Menasce D., Almeida V., Fonseca R., and Mendes M. Busniess-oriented Resource Management Policies for E-commerce Servers. *Performance Evaluation*, 42(2-3):223–239, 2000.

[157] Menasce D. A., Almeida V. A., Fonseca R., Marco A., and Mendes M. A. Resource Management Policies for E-commerce Servers. *ACM Performance Evaluation Review*, 27(4):27–35, 2000.

[158] Elnikety S., Tracey J., Nahum E., and Zwaenepoel W. A Method for Transparent Admission Control and Request Scheduling in E-comemrce Web Sites. In *ACM World Wide Web Conference*, pages 276–286, New York, U.S., May 2004.

[159] Nichols K., Blake S., Baker F., and Black D. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. Technical report, RFC 2474, Dec 1998.

[160] Jacobson V. and Karels M. J. Congestion Avoidance and Control. In *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 314–329, California, U.S., Aug. 1988.

[161] Gevros P. and Crowcroft J. Experimental Results on Weighted Proportional TCP Throughput Differentiation. In *International Workshop on High Performance Protocol Architectures*, London, U.K., June 1998.

[162] Floyd S. A Report on Some Recent Developments in TCP Congestion Control. *IEEE Communications Magazine*, 39(4):84–90, April 2001.

[163] Elek V., Karlsson G., and Ronngren R. Admission Control Based on End-to-End Measurements. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 623–630, Tel Aviv, Israel, 2000.

[164] Qiu J. and Knightly W. Measurement-based Admission Control with Aggregate Traffic Envelopes. *IEEE/ACM Transactions on Networking*, 9(2):199–210, April 2001.

[165] Jamin S., Danzig P. B., Shenker S. J., and Zhang L. A Measurement-based Admission Control Algorithm for Integrated Service Packet Networks. *IEEE/ACM Transactions on Networking*, 5(1):56–70, Feb 1997.

[166] Breslau L., Knightly E. W., Shenker S., Stoica I., and Zhang H. Endpoint Admission Control: Architectural Issues and Performance. In *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 57–69, Stockholm, Sweden, 2000.

[167] Breslau L. and Jamin S. Comments on the Performance of Measurement-based Admission Control Algorithms. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1233–1242, Tel Aviv, Israel, 2000.

[168] Cocchi R., Estrin D., Shenker S., and Zhang L. A Study of Priority Pricing in Multiple Class Networks. In *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 123–130, Zurich, Switzerland, Sept 1991.

[169] Mackie-Mason J. K. and Varian H. Pricing Congestible Network Resources. *IEEE Journal of Selected Areas in Communications*, 13(7):1141–1149, Sept. 1995.

[170] Mackie-Mason J. K. and Varian H. Pricing the Internet. In *Public Access to the Internet, B. Kahin and J. Keller, Editors.* MIT Press, 1995.

[171] Mackie-Mason J. K., Murphy L., and Murphy J. The Role of Responsive Pricing in the Internet. In *The Internet, Internet Economics, J. Bailey and L. McKnight Editors.* MIT Press, 1997.

[172] Gibbens R. and Kelly F. Resource Pricing and the Evolution of Congestion Control. *Automatica*, 35:1969–1985, 1999.

[173] Shenker S., Clark D., Estrin D., and Herzog S. Pricing in Computer Networks: Reshaping the Research Agenda. *ACM Computer Communication Review*, 26(2):19–43, 1996.

[174] Wang X. and Schulzrinne H. Performance Study of Congestion Price Based Adaptive Service. In *International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 1–10, North Carolina, U.S., June 2000.

[175] Varian H. *Microeconomics Analysis.* W.W.Norton & Co, 1993.

[176] Odlyzko A. Paris Metro Pricing for the Internet. In *ACM Conference on Electronic Commerce*, pages 140–147, Colorado, U.S., 1999.

[177] Leyton-Brown K., Porter R., Venkataraman S., and Prabhakar B. Smoothing out Focused Demand for Network Resources. In *ACM Electronic Commerce Conference*, pages 245–248, Florida, U.S., Oct 2001.

[178] Abdelzaher T. F. and Lu Chenyang. Schedulability Analysis and Utilization Bounds for Highly Scalable Real-Time Services. In *IEEE Real-time Technology and Applications Symposium*, pages 15–25, Taiwan, 2001.

Appendix A

ECBench Database Table Description

Field	*Type*	*Description*
I_ID	Numeric, 10 digits	Unique ID of item
I_TITLE	Variable text, size 60	Title of item
I_A_ID	Numeric, 10 digits	Author ID of item
I_PUB_DATE	Date	Date of release of the product
I_PUBLISHER	Variable text, size 60	Publisher of item
I_SUBJECT	Variable text, size 60	Subject of book
I_DESC	Variable text, size 600	Description of item
I_THUMBNAIL	Image	Thumbnail image of item
I_SRP	Numeric, (15,2) digits	Suggested retail price
I_COST	Numeric, (15,2) digits	Cost of item
I_AVAIL	Date	When item is available
I_STOCK	Numeric, 4 digits	Quantity in stock
I_ISBN	Fixed text, size 13	Product ISBN
I_PAGE	Numeric, 4 digits	Number of pages of book
I_BACKING	Variable text, size 15	Type of book, paper or hard back
I_DIMENSIONS	Variable text, size 25	Size of book in inches

Table A.1: Item Table Fields Description [3]

Field	*Type*	*Description*
A_ID	Numeric, 10 digits	Unique author ID
A_FNAME	Variable text, size 20	First name of author
A_LNAME	Variable text, size 20	Last name of author
A_MNAME	Variable text, size 20	Middle name of author
A_DOB	Date	Date of birth of Author
A_BIO	Variable text, size 500	About the author

Table A.2: Author Table Fields Description [3]

Field	*Type*	*Description*
C_ID	Numeric, 10 digits	Unique ID per customer
C_UNAME	Variable text, size 20	Unique user name for customer
C_PASSWD	Variable text, size 20	User password for customer
C_FNAME	Variable text, size 15	First name of customer
C_LNAME	Variable text, size 15	Last name of customer
C_ADDR_ID	Numeric, 10 digits	Address ID for customer
C_PHONE	Variable text, size 16	Phone number of customer
C_EMAIL	Variable text, size 50	For sending purchase information
C_SINCE	Date	Date of customer registration
C_LAST_VISIT	Date	Date of last visit
C_LOGIN	Date and time	Start of current customer session
C_EXPIRATION	Date and time	Current customer session expiry
C_DISCOUNT	Numeric, (3,2) digits	Percentage discount for customer
C_BALANCE	Sign numeric, (15,2) digits	Payment of customer
C_BIRTHDATE	Date	Birth date of customer
C_DATA	Variable text, size 500	Miscellaneous information

Table A.3: Customer Table Fields Description [3]

Field	*Type*	*Description*
O_ID	Numeric, 10 digits	Unique ID per order
O_C_ID	Numeric, 10 digits	Customer ID of order
O_DATE	Date and time	Order date and time
O_SUB_TOTAL	Numeric, (15,2) digits	Subtotal of all order-line items
O_TAX	Numeric, (15,2) digits	Tax over the subtotal
O_SHIP_TYPE	Variable text, size 10	Method of delivery
O_SHIP_DATE	Date and time	Order ship date
O_BILL_ADDR_ID	Numeric, 10 digits	Address ID to bill
O_SHIP_ADDR_ID	Numeric, 10 digits	Address ID to ship order
O_STATUS	Variable text, size 15	Order status

Table A.4: Orders Table Fields Description [3]

Field	*Type*	*Description*
OL_ID	Numeric, 3 digits	Unique order line item ID
OL_O_ID	Numeric, 10 digits	Order ID of order line
OL_I_ID	Numeric, 10 digits	Unique item ID (I_ID)
OL_QTY	Numeric, 3 digits	Quantity of item
OL_DISCOUNT	Numeric, (3,2) digits	Percentage discount off of I_SRP
OL_COMMENTS	Variable text, size 100	Special instructions

Table A.5: Order_line Table Fields Description [3]

Field	*Type*	*Description*
ADDR_ID	Numeric, 10 digits	Unique address ID
ADDR_STREET1	Variable text, size 40	Street address, line 1
ADDR_STREET2	Variable text, size 40	Street address, line 2
ADDR_CITY	Variable text, size 30	Name of city
ADDR_STATE	Variable text, size 20	Name of state
ADDR_ZIP	Variable text, size 10	Zip code or postal code
ADDR_CO_ID	Numeric, 4 digits	Unique ID of country

Table A.6: Address Table Fields Description [3]

Field	*Type*	*Description*
CO_ID	Numeric, 4 digits	Unique country Id
CO_NAME	Variable text, size 50	Name of country
CO_EXCHANGE	Numeric, (6,6) digits	Exchange rate to US Dollars
CO_CURRENCY	Variable text, size 18	Name of currency

Table A.7: Country Table Fields Description [3]

Field	*Type*	*Description*
CX_O_ID	Numeric, 10 digits	Unique order ID (O_ID)
CX_TYPE	Variable text, size 10	Credit card type
CX_NUM	Numeric, 16 digits	Credit card number
CX_NAME	Variable text, size 31	Name on credit card
CX_EXPIRY	Date	Expiration date of credit card
CX_AUTH_ID	Fixed text, size 15	Authorisation for transaction amount
CX_XACT_AMT	Numeric, (15,2) digits	Amount for this transaction
CX_XACT_DATE	Date and time	Date and time of authorisation
CX_CO_ID	Numeric, 4 digits	Country where transaction originated

Table A.8: CC_Xacts Table Fields Description [3]

Appendix B

TPC-W Random Methods for Database Population

Field	*Description*
a-string	Generates a string using an independent uniform random distribution drawn from all letters, numerical values and special characters.
n-string	Generates a string of numeric characters using an independent uniform random distribution drawn from the set of all numeric digits.
random(x,y)	Generates a random value independently selected and uniformly distributed between x and y.
randomPermutation(x,y)	Generates a sequence of numbers from x to y arranged into a random order.
DigSyl(D,N)	Returns a string which is the concatenation of 2-character syllables constructed by replacing each digit in the decimal representation of D with the corresponding 2-character syllable shown in Table B.2.

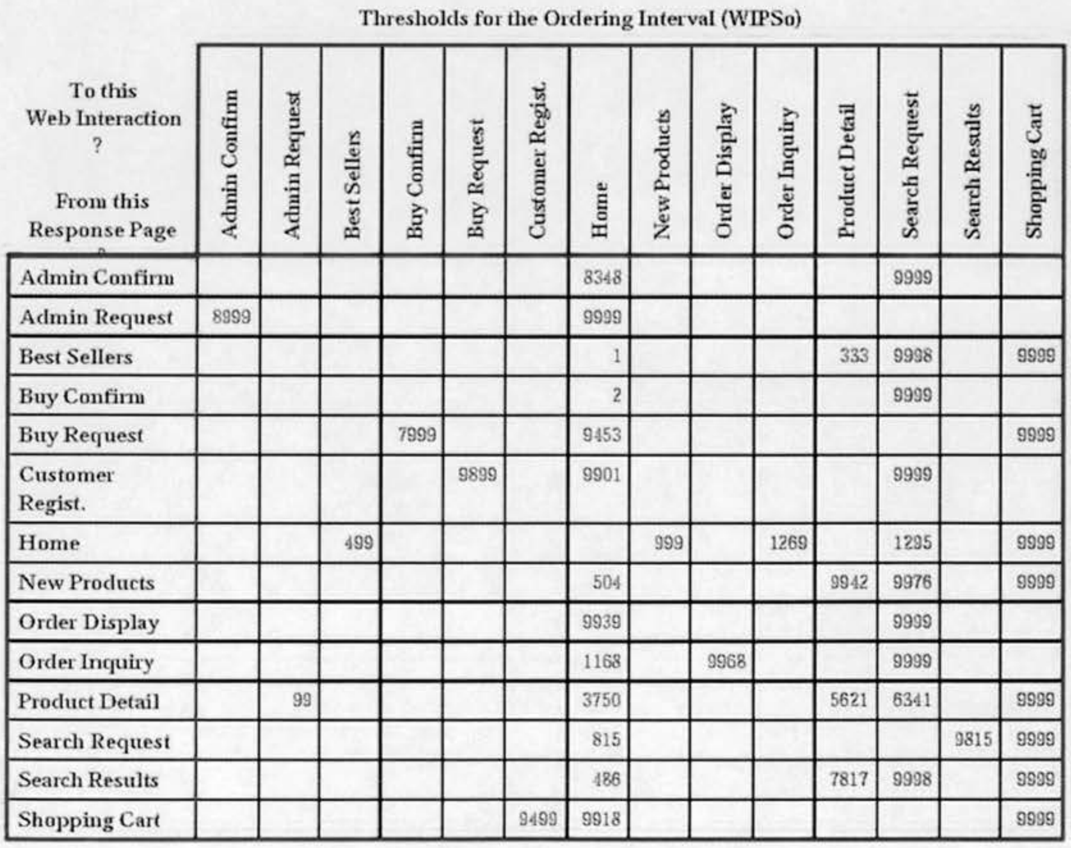
Table B.1: RandomMethods Library's Random Functions, Based on TPC-W [3]

Digit	0	1	2	3	4	5	6	7	8	9
Syllable	BA	OG	AL	RI	RE	SE	AT	UL	IN	NG

Table B.2: DigSyl Syllable Table, Based on TPC-W [3]

Appendix C

TPC-W's Navigational Pattern Thresholds for an Ordering Interval



Thresholds for the Ordering Interval (WIPSo)

To this Web Interaction ? / From this Response Page ?	Admin Confirm	Admin Request	Best Sellers	Buy Confirm	Buy Request	Customer Regist.	Home	New Products	Order Display	Order Inquiry	Product Detail	Search Request	Search Results	Shopping Cart
Admin Confirm							8348					9999		
Admin Request	8999						9999							
Best Sellers							1				333	9998		9999
Buy Confirm							2					9999		
Buy Request				7999			9453							9999
Customer Regist.					9899		9901					9999		
Home			499					999		1269		1295		9999
New Products							504				9942	9976		9999
Order Display							9939					9999		
Order Inquiry							1168		9968			9999		
Product Detail		99					3750				5621	6341		9999
Search Request							815						9815	9999
Search Results							486				7817	9998		9999
Shopping Cart						9499	9918							9999

Figure C.1: TPC-W's Thresholds for an Ordering Interval [3]

Appendix D

PHP and Servlets Benchmarking

Average response time results obtained, for all web interaction, when benchmarking PHP and Java Servlets application server implementations using ECBench.

Workload Size	*PHP*	*Java Servlets*
1	11.64	19.15
2	169.32	166.83
4	625.3	592.73
8	2366.5	2083.42
16	5401.1	5665.72
32	12935.5	11976.6
64	12486.9	13301.9
128	13008.9	10942.9
256	14511.9	13895.25
512	14079.8	19029.31
1024	14445.6	25214.72

Table D.1: Home Interaction Average Response Time (ms)

Workload Size	*PHP*	*Java Servlets*
1	344.5	451.8
2	535.7	855.5
4	1077.04	1653.95
8	2817.31	3336.42
16	7073.93	10734.32
32	13306.81	13654.9
64	14651.1	14302.38
128	11905.6	14273.1
256	15013.5	12093.22
512	14615.13	22365.61
1024	18281.14	40445.9

Table D.2: Best Sellers Interaction Average Response Time (ms)

Workload Size	*PHP*	*Java Servlets*
1	35.5	43.65
2	622.82	290.3
4	1233.83	589.1
8	3250.5	2522.2
16	6291.23	6588.7
32	12054.22	12853.65
64	15223.5	14132.9
128	13133.4	11115.8
256	11051.5	14350.3
512	14259.4	23677.1
1024	16191.1	37685.9

Table D.3: New Products Interaction Average Response Time (ms)

Workload Size	*PHP*	*Java Servlets*
1	7.6	17.1
2	141.1	184.1
4	497.33	561.1
8	1749.8	2105.2
16	5580.3	5275.63
32	12117.62	12188.42
64	11618.9	13291.7
128	12682.5	11272.6
256	11029.9	11869.1
512	10716.2	21049.0
1024	11450.43	29115.63

Table D.4: Search Request Interaction Average Response Time (ms)

Workload Size	*PHP*	*Java Servlets*
1	28.4	43.9
2	302.8	241.2
4	1174.15	665.4
8	2478.8	2203.75
16	5704.4	5533.65
32	12695.34	11864.8
64	13276.3	13335.4
128	12898.9	11515.8
256	11999.4	11925.4
512	14148.2	24400.21
1024	15174.7	31480.33

Table D.5: Search Result Interaction Average Response Time (ms)

Workload Size	*PHP*	*Java Servlets*
1	13.4	27.5
2	194.7	255.34
4	715.9	673.1
8	2212.92	2408.4
16	5265.43	5527.9
32	9610.44	9407.4
64	10238.2	10648.3
128	9414.6	9856.34
256	9759.2	10603.1
512	11069.31	25163.35
1024	12277.15	35283.42

Table D.6: Product Detail Interaction Average Response Time (ms)

Workload Size	*PHP*	*Java Servlets*
1	51.03	30.9
2	238.7	193.9
4	633.84	674.69
8	2034.5	2448.13
16	4896.12	4544.4
32	12641.1	12009.41
64	13785.4	13247.25
128	13348.5	11739.9
256	11138.65	12982.7
512	11345.2	24446.31
1024	13027.2	29868.32

Table D.7: Shopping Cart Interaction Average Response Time (ms)

Workload Size	*PHP*	*Java Servlets*
1	22.31	16.33
2	104.85	85.1
4	396.13	339.4
8	1176.92	1720.44
16	5063.4	4655.4
32	6840.7	7778.62
64	10225.7	7716.5
128	8332.1	6173.9
256	10906.8	9653.9
512	7633.1	15097.5
1024	9643.8	36080.94

Table D.8: Customer Registration Interaction Average Response Time (ms)

Workload Size	*PHP*	*Java Servlets*
1	234.6	251.9
2	423.13	360.71
4	973.53	826.1
8	1126.4	2162.93
16	2672.33	1971.26
32	4379.35	5593.13
64	6228.9	6010.2
128	6259.34	6909.22
256	6716.96	7673.54
512	7058.21	30333.68
1024	11633.24	30645.1

Table D.9: Buy Request Interaction Average Response Time (ms)

Workload Size	*PHP*	*Java Servlets*
1	249.66	61.8
2	266.38	213.95
4	666.13	404.6
8	1947.22	1940.36
16	1978.33	3021.94
32	6940.55	4015.35
64	4870.74	7001.18
128	6736.8	3586.11
256	6020.42	7605.5
512	10285.78	31231.0
1024	13749.11	32034.5

Table D.10: Buy Confirm Interaction Average Response Time (ms)

Workload Size	*PHP*	*Java Servlets*
1	9.42	22.1
2	340.16	124.0
4	20.4	675.66
8	1324.51	1642.97
16	1474.1	4338.27
32	7403.55	9073.1
64	7280.8	5766.77
128	12096.75	12277.13
256	8579.88	10923.0
512	11090.71	28933.14
1024	13978.36	31394.33

Table D.11: Order Inquiry Interaction Average Response Time (ms)

Workload Size	*PHP*	*Java Servlets*
1	4109.21	382.74
2	1900.0	431.9
4	1389.5	998.77
8	3165.88	2257.52
16	3678.44	4141.1
32	5703.0	7856.1
64	4664.2	5119.66
128	6470.5	9753.94
256	6346.2	7649.4
512	9529.93	25271.61
1024	16934.33	42915.33

Table D.12: Order Display Interaction Average Response Time (ms)

Appendix E

Service Time Analysis for *CBAC-disabled* and CBAC Experiments

	CBAC disabled		*CBAC1*		*CBAC2*		*CBAC3*		*CBAC4*	
Interaction	μ (s)	s	μ (s)	s	μ (s)	s	μ (s)	s	μ (s)	s
Home	6.68	18.65	0.27	1.05	0.43	1.29	0.55	1.56	0.47	1.43
Search Request	6.27	14.8	0.31	0.81	0.37	1.1	0.5	1.34	0.44	1.42
Search Result	5.85	12.68	0.31	0.8	0.37	0.88	0.51	1.26	0.43	1.16
Product Detail	5.96	17.82	0.32	0.79	0.36	0.88	0.48	1.2	0.39	0.94
Shopping Cart	5.4	5.88	0.3	0.89	0.37	0.98	0.47	1.2	0.41	1.02
Customer Reg.	6.1	17.7	0.32	0.84	0.39	1.03	0.48	1.18	0.4	1.03
Buy Request	7.11	19.84	0.86	0.97	0.87	1.04	1.03	1.24	0.92	1.1
Buy Confirm	6.17	14.4	0.34	0.77	0.41	0.9	0.53	1.14	0.45	0.98
Best Sellers	7.85	7.91	1.25	1.81	0.8	1.76	0.93	1.86	0.79	1.8
New Products	7.92	18.37	0.78	1.42	0.53	1.24	0.71	1.6	0.6	1.44

Table E.1: All Web Interactions Mean and Standard Deviation Service Time Analysis for CBAC-disabled and CBAC experiments